Ranking Support for Keyword Search on Structured Data using Relevance Models

Veli Bicer
FZI Forschungszentrum Informatik
Haid-und-Neu-Str. 10-14
76131, Karlsruhe, Germany
bicer@fzi.de

Thanh Tran
Institute AIFB
Geb. 11.40, KIT-Campus Süd
76128, Karlsruhe , Germany
thanh@kit.edu

Radoslav Nedkov
disy Informationssysteme GmbH
Erbprinzenstr. 4-12
76133, Karlsruhe, Germany
radoslav.nedkov@disy.net

ABSTRACT

Keyword query processing over structured data has gained a lot of interest as keywords have proven to be an intuitive mean for accessing complex results in databases. While there is a large body of work that provides different mechanisms for computing keyword search results efficiently, a recent study has shown that the problem of ranking is much neglected. Existing strategies employ heuristics that perform only in ad-hoc experiments but fail to consistently and repeatedly deliver results across different information needs. We provide a principled approach for ranking that focuses on a well-established notion of what constitutes relevant keyword search results. In particular, we adopt *relevance-based language models* to consider the structure and semantics of keyword search results, and introduce novel strategies for *smoothing* probabilities in this structured data setting. Using a standardized evaluation framework, we show that our work largely and consistently outperforms all existing systems across datasets and various information needs.

Categories and Subject Descriptors

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

General Terms

Theory,Algorithms

Keywords

Relevance Models, Keyword Search, Structured Data

1. INTRODUCTION

Keyword query processing over structured data has gained a lot of interest as keywords have proven to be an intuitive mean for accessing information and the amount of available structured data increases rapidly, especially in the realm of Web databases. Keyword search helps to circumvent the complexity of structured query languages, and hide the underlying data representation. Without knowledge of the query syntax and data schema, users can obtain complex structured results, including tuples from relational databases, XML data, data graphs, and RDF resources [1, 5, 3, 18]. As opposed to standard keyword search where retrieval units are single documents, results in this setting may encompass several resources that are connected over possibly very long paths (e.g. joined database tuples, XML trees, RDF resources connected over paths of relations).

Although much attention has been focused on finding efficient techniques to process keyword queries and to retrieve the structured data in these settings, only a small number of dedicated work can be found on the ranking of results and understanding their relevance to the user information need. One main direction is the use of Information Retrieval (IR) inspired TF-IDF ranking [11, 12]. Another direction is proximity search where the goal is to minimize the distance between data elements matching the keywords [1, 5]. For computing this weight of paths (distance), a PageRank-based metric is often incorporated for including the node prestige [5]. While high-quality results have been reported in the ad-hoc evaluations of these approaches, a recent study [2] that specifically focuses on benchmarking the effectiveness of keyword search ranking strategies has revealed serious problems: in contrast to previously published results, there is no ranking strategies that is clearly superior, and effectiveness results are much worst than those reported when using a principled approach for assessing relevance and a broader spectrum of queries.

The major shortcomings of previous work can be summarized as follows: the minimal distance heuristic behind proximity search is rather convenient for the efficient computation of results but does not directly capture relevance. The adoption of IR-based ranking proposed so far is also problematic because unlike document ranking, the score of a result in this setting is an aggregation of several resources' scores. Combining resources with high "local scores" however, does not always guarantee highly relevant final results [12]. Recent evaluation results [2] suggest that the proposed normalization methods [11, 12] are not effective in dealing with this issue. More importantly, previous work implicitly assumes that relevance is completely captured by the keyword query that is mostly short and ambiguous. We consider this assumption to be too strong especially in this

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

setting, where users might not be aware of the underlying structure and terminology of the database and thus, cannot specify "complete queries".

In this work, we make the following contributions: (1) we propose the use of Relevance Models (RM) for dealing with the ranking of structured results in keyword search. RM has been used for document ranking [8], which is a probabilistic model that directly captures the notion of relevance behind documents and queries. Primarily, it is employed as a form of query expansion, where relevance (and the query model) is constructed based on the set of documents obtained from an initial query run, i.e., via pseudo-relevance feedback (PRF). While it has shown to be effective in dealing with document ranking, the use of RM has not been studied in this keyword search setting before. (2) We adopt this model, with the goal of obtaining a representation more fine-grained than the original RM that can also exploit the structure of the PRF results. That is, instead of relying on the possibly short and ambiguous query, we use a model of relevance derived from both the content and structure of PRF results. (3) We introduce smoothing strategies that exploit the specific structure and semantics of the underlying data to obtain better estimates of probabilities that make up the RM. (4) We proposed a relevance-based ranking function that employs this adopted RM for ranking standalone as well as aggregated structured results. In particular, we show that this ranking function not only provides a principled way for the aggregation of "local scores", but also, it is monotonic. This is a crucial design issue because only monotonic functions are compatible with existing top-k techniques proposed for the efficient computation of keyword search results. (5) As opposed to previous ad-hoc experiments, we employ the general framework recently proposed for evaluating keyword search ranking strategies [2]. To the best of our knowledge, this is the first work that – based on a standardized evaluation – largely and consistently outperforms all existing systems across datasets and information needs in terms of precision, recall and MAP.

Structure. We introduce the readers to the problem of computing and ranking structured results for keyword queries in Section 2. Then, our approach for ranking is discussed in detail in Section 3. Section 4 contains results of the experiments. We discuss related work in Section 5 and conclude in Section 6.

2. KEYWORD SEARCH ON STRUCTURED DATA

In this section, we provide the definition of our keyword search problem an present an overview of existing approaches.

2.1 Problem Setting

Keyword search approaches have been proposed for dealing with different kinds of data, including relational, XML and RDF data. Generally speaking, the underlying data can be conceived as a directed labeled data graph $G = (V, E)$, where V is a disjoint union ($V = V_R \uplus V_A$) of resource nodes (V_R), and attribute nodes (V_A), and $E = E_F \uplus E_A$ represents a disjoint union of relation edges also called foreign key edges (E_F) that connect resource nodes, and attribute edges (E_A) that link between a resource node and an attribute node. In the following, we denote all attributes of the resource $r \in V_R$ as $\mathcal{A}(r)$, the attributes reachable from r via the edge e as $\mathcal{A}(r, e)$, and all the outgoing edges of r as $\mathcal{E}(r)$. This model closely resembles the graph-structured RDF data model (omitting special features such as RDF blank nodes). The intuitive mapping of this model to relational data is as follows: a database tuple captures a resource, its attributes, and references to related resources in the form of foreign keys; the column names correspond to edge labels. Example data and its corresponding data graph is shown in Figure 1.

The user query Q is a set of keywords $(q_1, ..., q_m)$. A keyword matches a resource node r if it matches any of the attribute nodes $\mathcal{A}(r)$, or any of the edges $\mathcal{E}(r)$. An answer to the user query Q is a minimal rooted directed tree that can be found in the data graph, which contains at least one matching resource for every keyword in Q. This is commonly referred to as a Steiner tree [1, 5]. In this work, we use the term *Joined Resource Tree* (JRT) to make clear that an answer is a joined set of resources represented as $JRT = \{r_1, \ldots, r_n\}$. In recent work, subgraphs have also been considered as keyword answers [18], instead of trees. While this difference in semantics requires more advanced mechanisms for result computation, we will show that this aspect is orthogonal to the problem of *keyword search result ranking* studied in this paper: given a user query, the goal here is to rank Steiner trees (or graphs) according to the user's perceived degrees of *relevance*. In other words, we want to produce a ranking of keyword search results that corresponds to the degree to which they *match the user information needs*. Unlike previous approaches which employ ad-hoc notions of relevance, and conducted different (and rather ad-hoc) experiments to assess this relevance, we aim to provide a principled approach to modeling and dealing with relevance, and will follow the general evaluation framework for evaluating ranking strategies as proposed recently [2].

2.2 Keyword Search Result Computation

Clearly, the main difference of keyword search on structured data and the traditional keyword search on documents is that instead of one single document, a Steiner tree may encompass several resources (e.g. database tuples, documents, RDF resource descriptions) that are connected over a possibly very long path of foreign key relationships.

There are *schema-based approaches* implemented on top of off-the-shelf databases ([4, 11, 12]). Basically, a keyword query is processed by mapping keywords to elements of the database. Then, candidate networks are computed from the schema, which represent valid join sequences (Steiner tree templates) that can be used to connect computed keyword elements. Formal structured queries are derived from candidate networks that finally, are evaluated using the underlying database engine. The main advantages of this approach is that the power and optimization capabilities of the underlying database engine can be fully utilized for computing structured results.

The *schema-agnostic approaches* ([5, 3, 10]) operate directly on the data. Since they do not rely on a schema, the applicability of these approaches is not limited to structured data. For instance, schema-agnostic approaches have been proposed for dealing with semi-structured RDF data [18] as well as the combination of structured, semi-structured and unstructured data [10]. Also here, keyword elements have to be identified first. Then, Steiner trees are iteratively com-

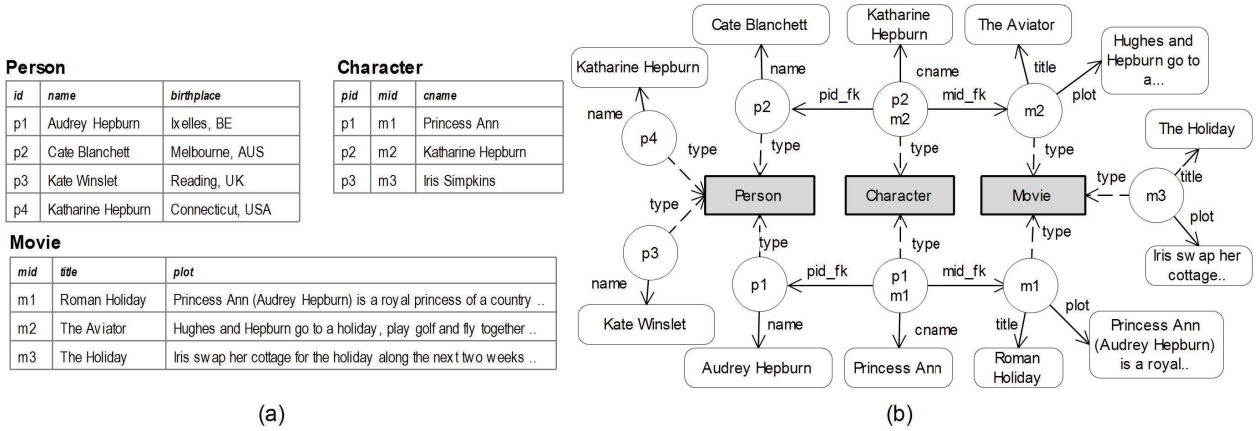

Figure 1: a) An example database from IMDB and b) its corresponding data graph (partially shown).

puted by exploring the underlying data graph that is mostly held in memory. For the query "Blanchett Aviator" for instance, this graph is simply the path between $p2$ and $m2$ in Fig. 1. Various kinds of algorithms have been proposed for the efficient exploration of data graphs, which might be very large (e.g. bidirectional search [5]).

While this large body of work on result computation is in principle orthogonal to the problem of ranking, there is one critical issue to be considered. Namely, existing approaches employ top-k processing techniques to focus only on the best results in order to terminate as early as possible. In particular, top-k processing terminates when the score of the k-best result obtained so far is guaranteed to be at least as high as the maximum score that can be achieved with the remaining candidates. The score of a result (i.e., a JRT), is typically defined as an aggregation of scores of the individual resources. The upper bound guarantee that is necessary for top-k can be ensured, when the overall score monotonically increases as more resources are added to the result during the process. In other words, existing top-k techniques assume the ranking function to be monotonic, a requirement we aim to satisfy so that our proposal can be used in combination with this previous work.

2.3 Keyword Search Result Ranking

The main idea behind ranking is to (1) assign each resource r in the JRT a score $Score(r)$, and then to combine the individual scores using a monotonic aggregation function $agg(\cdot)$ to obtain the final score $Score(JRT)$. Most commonly used is the IR-style ranking function adopted from TF-IDF based ranking. For instance, Discover [11] uses the sum of individual TF-IDF scores obtained via pivoted normalization weighting [16]:

$$\begin{aligned} Score(JRT) &= \sum_{r \in JRT} Score(r), \\ Score(r) &= \sum_{v \in r,Q} weight(v,r) * weight(v,Q), \end{aligned}$$

$$\begin{aligned} weight(v,r) &= \frac{ntf}{ndl} * idf, \\ ntf &= 1 + ln(1 + ln(tf)), \\ ndl &= (1-s) + s * \frac{dl}{avgdl}, \\ idf &= ln\frac{N}{df+1}, \end{aligned} \quad (1)$$

where ntf is the normalized term frequency (the normalized number of occurrences of v in r), df is the document frequency (the number of r containing the term v), N is the total number of resources in the collection, idf is the inverse document frequency, dl is the length measured as the number of terms contained in r, $avgdl$ is the average length in the collection, s is a constant usually set to 0.2, and ndl is the normalized document length.

In fact, Discover adopts this weighting via four different normalization methods. The goal is to obtain customized (1) idf and (2) ndl values, and to introduce (3) inter-document weight normalization as well as (4) Steiner tree size normalization. They are motivated by the facts that in this scenario, each column text has different vocabularies and thus shall be treated as a single collection, a Steiner tree is actually an "aggregated resource" and thus requires additional normalization beyond the resource level, and similar to document length, the size of the tree shall have an effect on relevance.

The last factor is very specific to this keyword search setting. Closely related to this Steiner tree size metric is the length of "root to matching nodes" paths [3], or "leaf to center nodes" paths [18]. All these metrics aim to capture the goal of what is known as proximity search, i.e., to minimize the distance between search terms (matching resource nodes) in the data graph. Applying this heuristic yields higher scores to those Steiner trees that are more compact. Intuitively speaking, the assumption here is that trees with more closely related matching nodes more likely capture the intended information need.

Besides IR-style scores defined for matching nodes (IR), and scores representing the distances between nodes in the result (proximity), the third category of scores commonly used is node prestige derived via PageRank [5]. Further,

while most scoring functions are designed to be monotonic, examples can be constructed where the resulting ranking contradicts human perception [12]. This gives rise to non-monotonic aggregation functions [12] that however, preclude the large body of existing query processing algorithms.

We note that these existing ranking strategies capture ad-hoc and often debatable intuitions of what supposed to be relevant. Traditionally, IR ranking (i.e., the probability ranking principle [19]) aims to maximize performance by ranking documents based on the posterior probability that they belong to the relevant class. That is, an explicit notion of relevance is employed, and specific approaches vary in their attempt to estimate word probabilities in this class. While the discussed TF-IDF normalizations [11] – and other existing IR-style ranking strategies [12] – intuitively make sense for the introduced examples, they lack a general account for relevance, i.e., it is not clear whether the resulting weights correspond to word probabilities in the relevant class. Also, while the distance-based heuristic used for proximity search is convenient from the computational point of view (as the keyword search problem can be reduced to the shortest-path problem [3]), it does not directly capture relevance in this sense.

An extensive evaluation of existing ranking strategies [2] suggests that no existing schemes is best for search effectiveness (contradicting previous ad-hoc evaluations). Proximity search that incorporates node prestige tends to be slightly more effective than IR-style ranking. Overall, the authors found that previously reported results were "abnormally well", which they attribute to non-standard relevance definitions coupled with very general queries. Further, non-monotonic ranking yields no appreciable difference such that the derived recommendation is "cheap ranking schemes should be used instead of complex ones that require completely new query processing algorithms". In light of these results, we will now present a principled approach to keyword search ranking that enables the reuse of out-of-the-box techniques for result computation.

3. RELEVANCE BASED RANKING

In this section, we present an IR-style ranking strategy. Instead of TF-IDF ranking and the ad-hoc normalization of weights proposed previously, we employ a principled method based on the use of language models that represent documents and queries as multinomial distributions over terms. In particular, we advocate the use of RM [8], aiming to directly capture the relevance behind the user query and documents.

3.1 Relevance Model

As a generic framework, RM is based on the generative relevance hypothesis that assumes for a given information need, queries and documents relevant to that need can be viewed as random samples from the same underlying generative model. Formally, an RM is defined as the function

$$RM_{\mathbf{R}}(v) = P(v \mid r_1, .., r_m) = \frac{P(v, r_1, .., r_m)}{P(r_1, .., r_m)}, \qquad (2)$$

where R captures the notion of relevance. The selection of R is highly critical for the effectiveness of the RM. Using a set F of PRF documents (i.e., documents obtained using the query Q) as an approximation of R, and assuming that query terms $q_i \in Q$ are independent, Lavrenko and Croft defined RM as

$$RM_F(v) \approx P(v \mid Q) = \sum_{D \in F} P(D) P(v \mid D) \prod_{q_i \in Q} P(q_i \mid D). \qquad (3)$$

Given a collection of documents C and the vocabulary of terms V, the score of a document $D \in C$ is based on its cross-entropy from the relevance model RM_F, defined as

$$H(RM_F \parallel D) = \sum_{v \in V} RM_F(v) \log P(v \mid D), \qquad (4)$$

where $P(v \mid D)$ is defined as

$$P(v \mid D) = \lambda_D \frac{n(v, D)}{\mid d \mid} + (1 - \lambda_D) P(v \mid C). \qquad (5)$$

Here, $n(v, D)$ is the count of the word v in the document, $\mid d \mid$ is the document length, and $P(v \mid C)$ is the background probability of v.

Intuitively speaking, relevance is modeled as a distribution over words obtained from PRF documents. This can be seen as a kind of query expansion. However, the difference is that instead of adding a few additional terms to the query, the relevance model here assigns a probability value to every term in the vocabulary V. Given the relevance and document as language models, cross-entropy is used as a similarity measure. Note that while constructing the document language model, the background collection probability is used for smoothing, which can be seen as a method for normalizing the document-specific probability of a term v.

3.2 Edge-specific Relevance Model

Given a user query $Q = (q_1, ..., q_m)$, we follow the RM approach to construct RM from a set of artifacts that is assumed to be close to the query. To achieve this, we use a *keyword index* over the data graph that index resources along with their attributes. Conceptually, this index can be seen as a query-resource map used for evaluating the multi-valued function $f : Q \rightarrow 2^{V_R}$. A standard inverted index is used for its implementation: every node $r \in V_R$ is treated as a document and document terms correspond to labels of attribute nodes $a \in \mathcal{A}(r)$. Further, we store edge labels in the index such that every term actually carries the information $(r, e, a) \in V_R \times E_A \times V_A$. After an initial run of the query Q over the keyword index, we obtain a PRF set of resources $F_R = \{r_1, ..., r_n\}$. Based on this, we construct an edge-specific relevance model (ERM) for each unique attribute edge e as

words	RM_q^{name}	$RM_q^{character}$	RM_q^{title}	RM_q^{plot}
hepburn	0.30	0.17	0.02	0.11
holiday	0.02	0.03	0.5	0.1
audrey	0.14	0.02	0.01	0.06
katharine	0.12	0.12	0.02	0.03
princess	0.01	0.01	0.02	0.1
roman	0.01	0.01	0.26	0.03
...	...	...	...	...

Table 1: Example ERM for the query "Hepburn Holiday"

$$RM^e_{F_R}(v) = \frac{\sum_{r \in F_R} P_{r \xrightarrow{e} a}(v \mid a) \prod_{i=1}^{m} P_{r \xrightarrow{*} a}(q_i \mid a)}{\sum_{r' \in F_R} \prod_{i=1}^{m} P_{r' \xrightarrow{*} a}(q_i \mid a)}, \qquad (6)$$

where $P_{r \xrightarrow{e} a}(v \mid a)$ represents the probability of observing a word v in the edge-specific attribute a (i.e., the attribute that is connected to r over e) and $P_{r \xrightarrow{*} a}(v \mid a)$ is the probability of observing the word in all the attributes of r. In fact, $P_{r \xrightarrow{*} a}(v \mid a)$ can be rewritten in the form of $P_{r \xrightarrow{e} a}(v \mid a)$ as

$$P_{r \xrightarrow{*} a}(v \mid a) = \sum_{e \in \mathcal{E}(r) \subseteq E_A} P_{r \xrightarrow{e} a}(v \mid a) P(e \mid r), \quad (7)$$

where $P(e \mid r)$ denotes the weight of edge e among all the edges of r. In particular, we estimate this by considering the length of the attributes of the edge as

$$P(e \mid r) = \frac{\sum_{a \in \mathcal{A}(r,e)} |a|}{\sum_{e' \in \mathcal{E}(r) \subseteq E_A} \sum_{a' \in \mathcal{A}(r,e')} |a'|}. \quad (8)$$

A trivial way to estimate the edge-specific attribute probabilities, $P_{r \xrightarrow{e} a}(v \mid a)$, is to consider every attribute as a document and to use a maximum likelihood estimation, $P^{ML}_{r \xrightarrow{e} a}(v \mid a)$, which is proportional to the count of the word v in an attribute a. However, such an estimation is problematic because it would assign zero probabilities to those words not occurring in the attribute. This may result in an underestimation of probabilities of the missing words. We will discuss later in Section 3.4 how to address this by applying a structure-based smoothing method to $P_{r \xrightarrow{e} a}(v \mid a)$.

Example. Consider the query "Hepburn Holiday" on the example data graph in Fig. 1. The keyword index returns the PRF resources $F_R = \{m1, p1, p4, m2, p2m2, m3\}$, each matches one or more query keywords. Based on this, we construct an ERM as illustrated in Table 1. Note the probabilities of the words vary for different edges. For example, "hepburn", and "audrey" are important words for the *name* or *plot* attributes, whereas "holiday" is given more emphasis as a movie *title*.

Compared to RM (equation 3), ERM (equation 6) takes the specific structures of the results into account. ERM is thus a more fine-grained approximation of relevance. It can be considered as analogous to form-based keyword search on databases in which the user is required to enter different keywords to different fields of the form. By processing the PRF set F_R of an initial query run, we are able to exploit the structure of the returned resources. This for instance, may capture attribute types such as *name*, *city*, *comment* etc. without any user intervention. Structures that can be incorporated might emerge from different resources with varying types of attributes.

3.3 Edge-Specific Resource Model

The basic retrieval unit here is a resource (a tuple) since the final results are obtained by combining resources (joining tuples) into an aggregated result. In order to calculate a final aggregated score and utilize an efficient top-k algorithm, we need to be able to assign each resource a score using our ranking mechanism. In order to achieve that, we construct a resource model that assign probabilities to the word in the vocabulary w.r.t. a given resource r. This is similar to the document model in standard IR approaches. The difference is that also structure information (the edges) is exploited for the estimation of word probabilities. We define an edge-specific resource model as

$$RM^e_r(v) = (1 - \lambda_r) P_{r \xrightarrow{e} a}(v \mid a) + \lambda_r P_{r \xrightarrow{*} a}(v \mid a). \quad (9)$$

Here, the probability of a word for a specific edge e of r is approximated as smoothed probabilities controlled by λ_r. Actually, the parameter λ_r is critical for our ranking to indicate the weight of the edge-specific attributes. A small λ_r means less smoothing and more emphasis on the edge-specific attribute (more emphasis on the terms of the attribute), and more emphasis on the terms of the entire resource (terms in all attributes) otherwise.

The score of a resource is then calculated based on the cross-entropy between the relevance model obtained for the query (RM_{F_R}) and the resource model (RM_r) as

$$Score(r) \quad = \quad \sum_{e \in E} \alpha_e \sum_{v \in V} RM^e_{F_R}(v) \log RM^e_r(v), \quad (10)$$

where α_e is a parameter that allow us to control the importance of different edges in the scoring.

3.4 Smoothing

Note that the core probabilities of both RM_{F_R} and RM_r is $P_{r \xrightarrow{e} a}(v \mid a)$. Smoothing is a well-known technique to address data spareness and improve the accuracy of language models, which we apply to obtain a better estimate for $P_{r \xrightarrow{e} a}(v \mid a)$. Traditional smoothing methods mainly use the global collection probabilities. In our case, the global collection simply comprises all attributes in the database. One deficiency of such a global smoothing is that it does not reflect the structure information that is available in this case. As an example, the smoothing probability of the word "Washington" in the *name* attribute "Denzel Washington" should not consider the probability of that word in the *city* attribute, since "Washington" can be highly frequent as a *city* but not as a *name*.

A general framework for smoothing is presented in [13], which consists of the tuple $\{S, f_u, \tilde{f}_u, w(u), w(u,v)\}$, where $S = (V_S, E_S)$ is the smoothing graph, f_u is the smoothed probabilities of the vertexes $u \in V_S$, $\tilde{f}_u$ is the non-smoothed (initial) probabilities of $u \in V_S$, $w(u)$ captures the weights indicating the importances of $u \in V_S$, and $w(u,v)$ stands for the weights of the edges $(u,v) \in E_S$.

Different instantiations of this framework result in different smoothing strategies. We adopt this by using the data graph G for S. The goal is to estimate the edge-specific attribute probabilities $f_u = P_{r \xrightarrow{e} a}(v \mid a)$. While $w(u)$ is set to be uniform, $\tilde{f}_u$ is computed via maximum likelihood as $P^{ML}_{r \xrightarrow{e} a}(v \mid a) = \frac{n(v,a)}{|a|}$, where the nominator is the count of v in a and the denominator denotes the length of a. Then, we obtain the smoothed probability

$$P_{r \xrightarrow{e} a}(v \mid a) = (1 - \lambda_a) P^{ML}_{r \xrightarrow{e} a}(v \mid a) + \lambda_a \sum_{a' \in \mathcal{N}(a)} \frac{w(a,a')}{Deg(a)} P_{r \xrightarrow{e} a}(v \mid a')$$

where $\mathcal{N}(a)$ is the neighborhood of a and:

$$Deg(a) = \sum_{a' \in \mathcal{N}(a)} w(a,a')$$

This notion of neighborhood in the case of structured data is illustrated in Fig. 2. More formally, given $a \in \mathcal{A}(r,e)$, a' is said to be in the neighborhood of a, $a' \in \mathcal{N}(a)$, iff:

- a and a' shares the same resources, i.e., $a' \in \mathcal{A}(r,e')$ for all edges $e' \in \mathcal{E}(r) - \{e\}$ (*Type 1*),
- resources of a and a' are of the same type, i.e., $a' \in \mathcal{A}(r',e)$ where $(r, type, c), (r', type, c) \in E$ (*Type 2*),

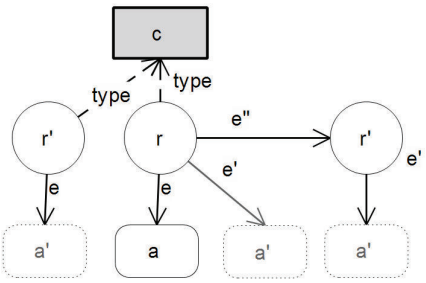

Figure 2: The neighborhood of attribute a.

- resources of a and a' are connected over a foreign key, i.e., $a' \in \mathcal{A}(r', e')$, where $(r, e'', r') \in E_F$ (*Type 3*).

The weight of these three types of attributes ($i \in \{1, 2, 3\}$) are determined separately as

$$w_i(a, a') = \frac{\gamma_i + \sigma(sim(a, a'))}{\gamma_i + 1} \quad \forall i \in \{1, 2, 3\} \tag{11}$$

Here, $\sigma(.)$ is sigmoid function, $sim(a, a')$ is the content-based similarity of the two attributes, and $\gamma_1, \gamma_2, \gamma_3$ are the control parameters for each type. A commonly used instantiation of $sim(a, a')$ is the cosine similarity that we also employ in our experiments.

3.5 Ranking

Ranking in this setting is concerned with JRT, which is a joined set of resources. We provided equation 10 for ranking individual resources. For ranking a set of resources, the same scoring function is applied. However, in this case, $RM^e_r(v)$ in equation 10 actually stands for the set of resources $r_m \in JRT$, defined as

$$RM^e_{JRT}(v) = \sqrt[m]{RM^e_{r_1}(v) \dots RM^e_{r_m}(v)}. \tag{12}$$

A critical issue for the efficient computation of results is that such an aggregated scoring function, which combines scores from more than two resources are monotonic:

Definition 1. (Score Monotonicity) *Let Q be the query, and $JRT = \{r_1, \dots, r_n\}$ and $JRT' = \{r'_1, \dots, r'_n\}$ be two results to Q. An aggregated scoring function is* monotonic *if it satisfies the following condition: if $Score(r_i) \leq Score(r'_i)$ for all $0 \leq i \leq n$, then $Score(JRT) \leq Score(JRT')$.*

We now show that the proposed ranking satisfies this:

Theorem 1. *The scoring function defined in equation 10 is monotonic with respect to the aggregation of resources defined in Equation 12.*

Proof:

$$\begin{aligned}
& Score(JRT) \\
= & \sum_{e \in E} \alpha_e \sum_v RM^e_{F_R}(v) \log \sqrt[m]{RM^e_{r_1}(v) \dots RM^e_{r_m}(v)} \\
= & \frac{1}{m} \sum_{e \in E} \alpha_e \sum_v RM^e_{F_R}(v) \left[\log RM^e_{r_1}(v) + \dots + \log RM^e_{r_m}(v)\right] \\
= & \frac{1}{m} \sum_{e \in E} \alpha_e \sum_v RM^e_{F_R}(v) \log RM^e_{r_1}(v) + \dots + \\
& \sum_{e \in E} \alpha_e \sum_v RM^e_{F_R}(v) \log RM^e_{r_m}(v) \\
= & \frac{1}{m}(Score(r_1) + \dots + Score(r_m))
\end{aligned}$$

Clearly, $\frac{1}{m}(Score(r_1) + \dots + Score(r_m))$ is monotonic. □

Example. Continuing with our example query "Hepburn Holiday", the keyword search algorithm firstly obtains the matching resources $F_R = \{m1, p1, p4, m2, p2m2, m3\}$. Based on this, several JRTs are constructed in a bottom-up top-k fashion. They are found based on exploring foreign key paths between these elements. Table 2 shows two of these paths (between p_1 and m_1, and between p_2m_2 and m_2) and two example JRTs resulting from these. During this computation, the scoring function defined in equation 10 is used both to compute the individual scores *Score(r)* (e.g. of p_1 and m_1) and *OurScore*, which indicates the combined score in Table 2 (e.g. of $\{p_2, m_2\} \rightarrow m_2$). Note the difference of our ranking scheme and the ones using proximity and TF-IDF. Proximity alone would assign $\{p_2, m_2\} \rightarrow m_2$ the highest score simply because it is the most compact answer. The two results tie in terms of TF-IDF scores. Our approach ranks $p_1 \leftarrow \{p_1, m_1\} \rightarrow m_1$ first, recognizing that "Hepburn" is an important term for *name* and "Holiday" is an important term for the *title* of movies.

4. EXPERIMENTS

Due to ad-hoc style evaluation, results of previous keyword search ranking were found to be abnormally well. Aiming at making results more conclusive and comparable, we exactly follow the framework for evaluating keyword search ranking strategies proposed recently [2]. For completeness, we will now summarize the data, queries and experimental settings proposed for this kind of evaluation.

4.1 Datasets

We use three different datasets, two of which are derived from two popular and large websites (Wikipedia and IMDb). IMDb data proposed for keyword search evaluation [2] is actually a subset from the original IMDb database, containing information about more than 180.000 movies. This is because several keyword search systems require data to be completely loaded into memory, and thus cannot scale to large datasets. The complete Wikipedia contains more than 3 milion articles. For the same reason, only a selection of more than 5500 articles was used. All tables unrelated to articles or users were excluded, and the PageLinks table was augmented with an additional foreign key to explicitly indicate referenced pages. The third dataset is MONDIAL, which represents the counterpoint to the other two because compared to them, it is smaller in size but more complex in terms of structure. It captures geographical and demographic information from the CIA World Factbook, the *International Atlas*, the TERRA database, and other web sources. The relational version for PostgreSQL was downloaded. Wikipedia contains more text than IMDb, which in turn, has more text then MONDIAL. In other words, while

JRT	$r \in JRT$	$tf_{hepburn}$	$tf_{holiday}$	$Score(r)$	$TF-IDF$	$Proximity$	Our Score
$p_1 \leftarrow \{p_1, m_1\} \rightarrow m_1$	p_1	1	0	0.36	3.0	3	0.22
	$\{p_1, m_1\}$	0	0	0.10			
	m_1	1	1	0.18			
$\{p_2, m_2\} \rightarrow m_2$	$\{p_2, m_2\}$	1	0	0.12	3.0	2	0.19
	m_2	1	1	0.26			

Table 2: Compare TF-IDF, proximity and our scores for the query "Hepburn Holiday".

MONDIAL can be seen as a structured database, Wikipedia is rather a structured document collection.

4.2 Queries

Clearly, performance may vary widely across information needs for the same document collection. Traditionally, fifty information needs are regarded as the minimum for evaluating retrieval systems. Accordingly, 50 queries were proposed [2] for each dataset to cover distinct information needs that vary in complexity. The maximum number of terms in any query is 7 and the average number of terms is 2.91. Among these different queries, there are two important types that were investigated in detail. There are the (1) *"TREC-style" queries* which are Wikipedia topics most similar to those encountered at TREC. Terms of these queries are present in many articles, yet most of those articles are not relevant. For instance, the query "smallpox vaccination" asks for information about the one who discovered/invented the smallpox vaccine. Finding out which articles are relevant and how they vary in the degree of relevance is the problem here. (2) The other type comprises *"single-resource"* queries which ask for exactly one resource. It has been reported that this type of queries constitute the most common type of query posed to existing search engines. For instance the query "rocky stallone" asks for the film in which the actor Sylvester Stallone plays the character Rocky.

4.3 Measuring the Degree of Relevance

Relevance is assessed based on the specified information needs. Binary relevance judgments are used such that all relevant results are equally desirable. Firstly, SQL queries are constructed to capture the information needs. Then, one single expert judges all results that can be obtained for these queries. Three metrics are employed to compare systems. (1) We use the number of *top-1 relevant results*, which is defined as the number of queries for which the first result is relevant. (2) *Reciprocal rank* is simply the reciprocal of the highest ranked relevant result for a given query. These measures aim to capture the quality of the top-ranked results. As the third metric, we use *average precision*, which also takes the order into account. For a query, it is defined as the average of the precision values calculated after each relevant result is retrieved (and assigning a precision of 0.0 to any relevant results not retrieved). Mean average precision (MAP) averages this single value across queries to obtain one single measure across different levels of recall and different types of information needs. These metrics are calculated based on the top-50 results returned by each system.

Table 3 provides a summary of the statistics of the data, the query workload, and the relevant results for each dataset.

4.4 Baseline Systems

We compare the results against systems, which have shown to provide best results in the previous evaluation [2]. There are *BANKS* [1] and *Bidirectional* [5], which represent the category of ranking strategies that make use of proximity and node prestige. The IR-style (TF-IDF based) ranking is implemented by *Efficient* [4], *SPARK* [12] and Covered Density [2] (*CD*). *SPARK* is the one that features a non-monotonic function. It has been found [2] that for single-resource queries, proximity in combination with node prestige perform well (compared to IR-style ranking). This is because this scheme prefers the smallest result that satisfies the query (i.e., it prefers less complex JRTs referring to only one single resource) while IR-style ranking scheme prefers larger results that contain additional instances of the search terms. *BANKS* and the like are the best ones on the Mondial dataset, outperforms the IR approaches on the IMDb dataset, and tie for the second most effective on Wikipedia. While this type of systems also provides reasonable effectiveness for the TREC-style queries, they are outperformed by systems implementing IR-style ranking. In particular, *Efficient* shows the best MAP result among all systems – for the set of Wikipedia topics used in the experiment. Further, SPARK reported results were not supported by previous finding, suggesting that the use of non-monotonic ranking function may not be beneficial after all.

Dataset	Size	Rel.	Tuples	|Q|	$\|\bar{q}\|$	$\|\bar{R}\|$
Mondial	9	28	17,115	50	2.04	5.90
IMDb	516	6	1,673,074	50	3.88	4.32
Wikipedia	550	6	206,318	50	2.66	3.26

Table 3: Characteristics of the three datasets and query workload. Size in MB, number of relations and tuples, total number of queries |Q|, average number of terms per query $|\bar{q}|$, and average number of relevant results per query $|\bar{R}|$.

4.5 Results

The goal of the experiment is to find out how the proposed ranking (*RM-S*) compares to the best systems on TREC-style as well as single-resource queries. We will firstly discuss the overall results, and then investigate these two types of queries in more detail.

4.5.1 Overall Results

Figure 3 summarizes the overall effectiveness of each system across different information needs in terms of MAP values. We see that the overall effectiveness varies considerably across the three datasets. On average, the best three systems are *Bidirectional*, *CD* and *RM-S*. In particular, our proposed ranking *RM-S* shows very convincing results. It consistently outperforms all systems across datasets. MAP values are above 0.78 and for the MONDIAL dataset, it is even over 0.9. For the IMDb dataset, MAP is close to 0.3

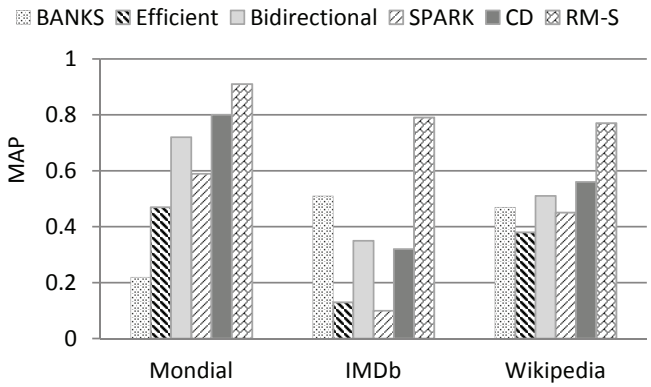

Figure 3: MAP across systems and datasets.

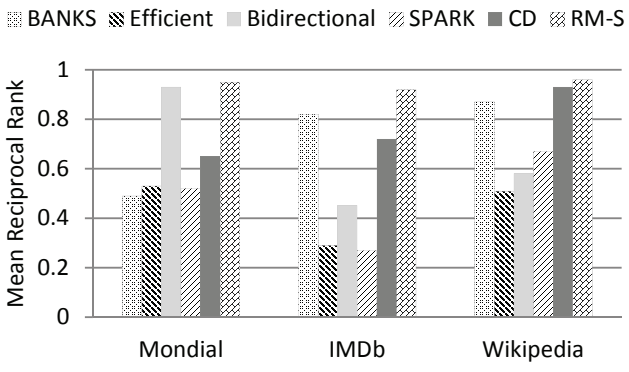

Figure 4: Reciprocal rank for single-resource queries.

higher then the second best system. These results are very encouraging, given the experiments have been conducted in a standardized way using real world datasets and a large set of queries. Also very important for practical purposes is the fact that the best systems including *RM-S* do not require non-monotonic ranking (as advocated by SPARK), and thus can be used in combination with state-of-the-art approaches for the efficient computation of keyword search results.

4.5.2 Single-Resource Queries

Figure 4 shows the mean reciprocal rank for each system for queries where exactly one resource is relevant. Performance of systems vary for this type of queries. As already reported in the previous study, *Bidirectional* and *BANKS* perform relatively well, outperforming the standard TF-IDF based systems such as *Efficient* and *SPARK* on average. *BANKS* achieves poor performance on the IMDb dataset but very good performance on the MONDIAL dataset, while *Bidirectional* exhibits more consistent performance. Our approach clearly shows best performance. It outperforms other approaches across all datasets, with mean reciprocal rank values being consistently above 0.91. We found out that TF-IDF based ranking tends to prefer complex results, which contain a large number of mentions of query terms. Thus, there are high rank results, which contain a large number of resources (each possibly containing mentions of several query terms). These results are however not relevant in this case because the queries target a single resource. In particular, the JRT size normalization (inspired by the document length normalization used in TF-IDF based ranking) [11] introduced to address this issue seems to be not as effective[1] as the more aggressive proximity-based scheme [1], which simply focuses on minimizing the tree size and finding compact results.

In light of these arguments, it seems surprising that *RM-S*, which does not incorporate this tree-size heuristic at all, achieves best results even in this case. Based on the query and the PRF results, it constructs a model of relevance and performs ranking entirely based on this model. Thus, while

[1]The performance of [11] is not shown because it is similar to *Efficient* in concept but worse in performance. It is also based on TF-IDF but employes additional normalization strategies to accommodate for differences in the keyword search setting.

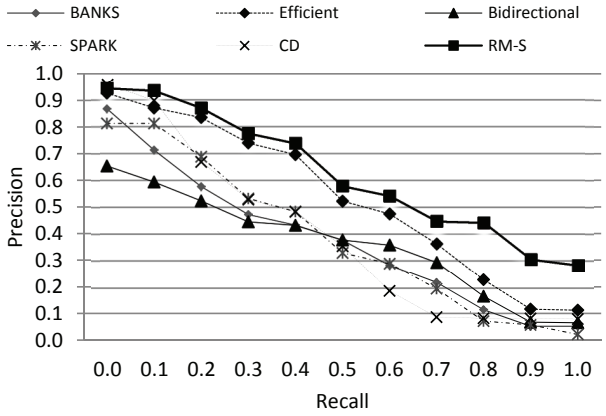

Figure 5: Precision-recall for TREC-style queries on Wikipedia.

RM-S can accommodate for and exploit the possibly varying structures of the PRF results, it does not directly assume that more compact results are necessary better. We believe the key here is that *RM-S* is able to make better estimates of the structure (i.e., the set of attributes) and the content (i.e., the terms) that make up relevant results (independent of their size). For instance for the IMDb query "rocky stallone", TF-IDF based ranking finds many movies but the one with Silvester Stallone as actor and Rocky as character is not the highest ranked one because the term "rocky" and "stallone" appear more frequently in the other movies. Also here, proximity based ranking fails because there is one movie with a character name that matches "rocky stallone". This one is ranked best because its JRT size is smaller compared to the JRT of the result with Rocky as character and Stallone as actor. In this case, *RM-S* is able to find PRF results where "stallone" appears as term in the attribute *actor* and "rocky" appears in the attribute *character*. Based on the resulting ERM, it successfully makes the guess that relevant results should have the attribute *actor* with terms such as "stallone" and "sylvester", and the attribute *character* with terms like "rocky" and "balboa".

4.5.3 TREC Queries

The results for this type of queries are shown in Figure

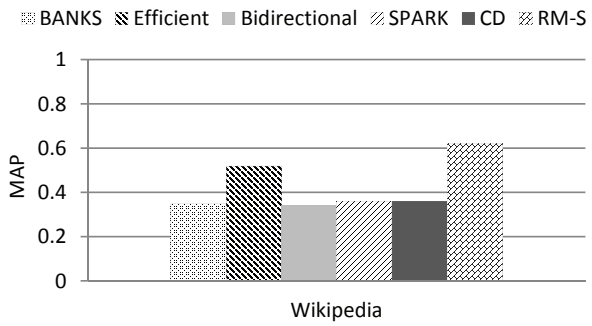

Figure 6: MAP for TREC-style queries on Wikipedia.

5 and Figure 6. While Figure 6 shows the overall performance, Figure 5 provides a breakdown into different levels of precision and recall. These results suggest that ranking schemes based on proximity and prestige do not perform well. *BANKS* is the best one in this category with precision close to 0.9 at the lowest recall level. Its precision however drops precipitously at higher recall levels. Expectedly, TF-IDF ranking performs better, with *Efficient* being the one with best performance in terms of MAP, and most stable performance across the entire precision-recall curve. In this category, *SPARK* performs best at low levels of recall. However, its precision drops sharply as recall increases over 0.1.

Also for this type of queries, our approach yields good results, providing best MAP (0.62) and stable performance over the entire precision-recall curve. It closely matches the performance of *SPARK* at the lowest level of recall, and consistently outperforms all approaches at higher recall levels. Up to recall level of 0.7, its performance is similar to the one of *Efficient*. It however provides much more stable performance at recall levels above that. In fact, we expect our approach to provide better recall (at the same level of precision) because using the relevance model is conceptually similar to query expansion. Terms and attributes, which make up relevant results are not limited to the ones specified in the query. Thus, relevant results can be determined even when they do exactly contain the query terms. For the query "smallpox vaccination" for instance, the resulting relevance model also contains terms such as "louis", "pasteur", "1794" and "virus". Surprising is however the fact that this does not come at the expense of precision. We believe that this is due to the fine-grained structure of the relevance model we employ, which reduces noises in the query expansion process. For instance, results are only deemed relevant when they contain the term "louis" in the attribute *inventor*.

4.5.4 Parameters

There is a number of parameters that we set experimentally based on the effectiveness of the ranking results. In particular, the PRF set of documents is $\mid F_R \mid \in \{5, 10, 25, 50, 75\}$, and the smoothing parameters for the edge-specific attribute properties $P_{r \xrightarrow{e} a}(v \mid a)$ are $\lambda_r \in \{0.1, 0.2, ..., 0.9\}$ and $\lambda_a \in \{0.1, 0.2, ..., 0.9\}$. We sweep over values for finding the best configuration of these parameters. Further, α_e is set to be uniform in all the experiments.

We analyze the sensitivity of MAP w.r.t. the smoothing parameter λ_a and control parameters $\gamma_1, \gamma_2, \gamma_3$ of Equation

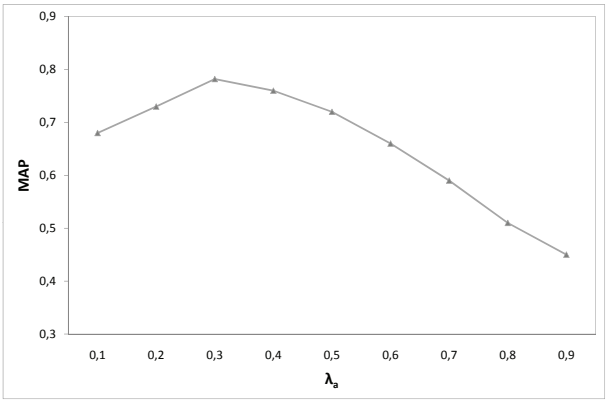

Figure 7: Sensitivity to smoothing interpolation parameter λ_a on Wikipedia.

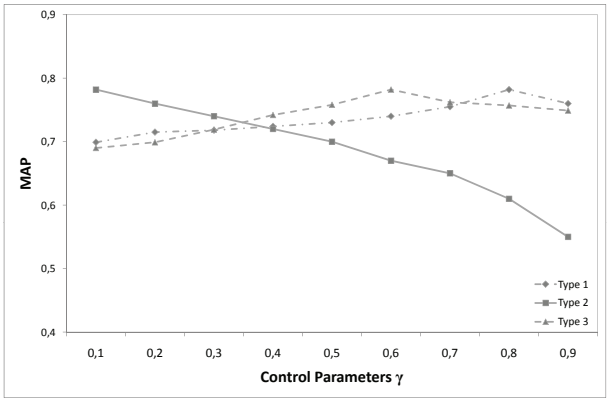

Figure 8: Sensitivity to control parameters of smoothing on Wikipedia ($\lambda_a = 0.3$).

11. We first fix the control parameters and show in Figure 7 how MAP changes according to the value of λ_a for the Wikipedia dataset. We see that MAP values are relatively high at low levels of smoothing, i.e., are best for λ_a in the range between 0.2 and 0.5. This means that while smoothing based on the structure of the data helps to improve performance, it should not be overemphasized. For a fixed $\lambda_a = 0.3$, we investigate the sensitivity of each control parameter by fixing the other two (based on best performance such that fixed values are $\gamma 1 = 0.8, \gamma 2 = 0.1$ and $\gamma 3 = 0.6$). As illustrated in Fig. 8, best performance can be achieved when more emphasis is put on Type 1 and Type 3 and less on Type 2.

5. RELATED WORK

Finding and ranking relevant resources is the core problem in the Information Retrieval community, for which different approaches have been investigated. The model we use here originates from the concept of language models [15], which have been proposed for modeling resources and queries as multinomial distributions over vocabulary terms, and for ranking based on the distance of the two models (e.g. using KL-divergence [20] or cross entropy [8] as measures for distance). More precisely, the foundation of our work is established by Lavrenko et al., who propose relevance-based language models to directly capture the relevance behind

document and queries. Further, the structure of results as well as queries have been incorporated into language models. For instance, combinations of language models constructed from fields of documents have been proposed for structured document retrieval [21], and combinations of attribute-level language models have been employed for the retrieval of complex Web objects [14] and semistructured XML data [6]. Also, structure information has been exploited for constructing structured relevance models [9] (SRM). This is the one mostly related to ERM. The difference is that while SRM consists of models derived from the structure specified in the query, ERM is derived from results obtained from unstructured keyword search. Also, the goal of SRM is to predict values of empty fields, whereas ERM targets the ranking of keyword search on structured data. In this setting, scores have to be combined from several resources (tuples), using a monotonic aggregation function. Thus, while ERM is similar to SRM in concept, the way it is constructed and how it is used are different.

The smoothing method we use represents an instantiation of an existing framework [13]. We adopt it to the case of structured data and show how the local neighborhood of resources can be exploited to obtain smoothed estimations of the edge-specific attribute probabilities. This technique is clearly different to existing work that instead, uses the local corpus structure of documents (e.g. document clusters, neighbors etc.) [7, 17].

Throughout the paper, we discussed existing approaches including various adoptions of IR-style ranking [11, 12] that focus on the problem of keyword search on structured data. However, we note that this is the first work that investigates the use of language models. Instead of employing questionable normalizations and heuristics, we provide a principled approach that directly models the relevance behind queries and results.

6. CONCLUSION

Keyword search on structured data is a popular problem for which various solutions exist. We focus on the aspect of keyword search result ranking, providing a principled approach that employs language models to capture results, queries and the relevance behind them. A recent study has shown that existing heuristics and normalizations proposed for this problem exhibit good results only in the previous adhoc experiments, but fail to deliver consistent performance across different information needs and datasets, and especially, do not deliver stable performance across the precision-recall curve (low precision at higher recall levels). Through a standardized evaluation, we show that our approach delivers superior results, largely outperforming all existing systems in terms of precision, recall and MAP. Further, we formally show that the ranking function is monotonic. This is of great value in practice, enabling the proposed ranking scheme to be used in combination with state of the art approaches for the efficient computation of results.

7. REFERENCES

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, and S. Sudarshan. Banks: Browsing and keyword searching in relational databases. In *VLDB*, pages 1083–1086, 2002.

[2] J. Coffman and A. C. Weaver. A framework for evaluating database keyword search strategies. In *CIKM*, pages 729–738, 2010.

[3] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD Conference*, pages 305–316, 2007.

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

[5] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.

[6] J. Kim, X. Xue, and W. B. Croft. A probabilistic retrieval model for semistructured data. In *ECIR*, pages 228–239, 2009.

[7] O. Kurland and L. Lee. Corpus structure, language models, and ad hoc information retrieval. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 194–201. ACM, 2004.

[8] V. Lavrenko and W. B. Croft. Relevance-based language models. In *SIGIR*, pages 120–127, 2001.

[9] V. Lavrenko, X. Yi, and J. Allan. Information retrieval on empty fields. In *HLT-NAACL*, pages 89–96, 2007.

[10] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD Conference*, pages 903–914, 2008.

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

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

[13] Q. Mei, D. Zhang, and C. Zhai. A general optimization framework for smoothing language models on graph structures. In *SIGIR*, pages 611–618, 2008.

[14] Z. Nie, Y. Ma, S. Shi, J.-R. Wen, and W.-Y. Ma. Web object retrieval. In *WWW*, pages 81–90, 2007.

[15] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *SIGIR*, pages 275–281, 1998.

[16] A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *SIGIR*, pages 21–29, 1996.

[17] T. Tao, X. Wang, Q. Mei, and C. Zhai. Language model information retrieval with document expansion. In *HLT-NAACL*, 2006.

[18] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE*, pages 405–416, 2009.

[19] M. Wechsler and P. Schäuble. The probability ranking principle revisited. *Inf. Retr.*, 3(3):217–227, 2000.

[20] C. Zhai and J. D. Lafferty. A risk minimization framework for information retrieval. *Inf. Process. Manage.*, 42(1):31–55, 2006.

[21] L. Zhao and J. Callan. A generative retrieval model for structured documents. In *CIKM*, pages 1163–1172, 2008.