Keyword Query Routing

Thanh Tran and Lei Zhang

Abstract—Keyword search is an intuitive paradigm for searching linked data sources on the web. We propose to route keywords only to relevant sources to reduce the high cost of processing keyword search queries over all sources. We propose a novel method for computing *top-k routing plans* based on their potentials to contain results for a given keyword query. We employ a *keyword-element relationship* summary that compactly represents relationships between keywords and the data elements mentioning them. A *multilevel scoring mechanism* is proposed for computing the relevance of routing plans based on scores at the level of keywords, data elements, element sets, and subgraphs that connect these elements. Experiments carried out using 150 publicly available sources on the web showed that valid plans (precision@1 of 0.92) that are highly relevant (mean reciprocal rank of 0.89) can be computed in 1 second on average on a single PC. Further, we show routing greatly helps to improve the performance of keyword search, without compromising its result quality.

Index Terms—Keyword search, keyword query, keyword query routing, graph-structured data, RDF

1 Introduction

The web is no longer only a collection of textual documents but also a web of interlinked data sources (e.g., Linked Data). One prominent project that largely contributes to this development is Linking Open Data. Through this project, a large amount of legacy data have been transformed to RDF, linked with other sources, and published as *Linked Data*. Collectively, Linked Data comprise hundreds of sources containing billions of RDF triples, which are connected by millions of links (see LOD Cloud illustration at http://linkeddata.org/). While different kinds of links can be established, the ones frequently published are *sameAs* links, which denote that two RDF resources represent the same real-world object. A sample of Linked Data on the web is illustrated in Fig. 1.

It is difficult for the typical web users to exploit this web data by means of structured queries using languages like SQL or SPARQL. To this end, keyword search has proven to be intuitive. As opposed to structured queries, no knowledge of the query language, the schema or the underlying data are needed.

In database research, solutions have been proposed, which given a keyword query, retrieve the most relevant structured results [1], [2], [3], [4], [5], or simply, select the single most relevant databases [6], [7]. However, these approaches are single-source solutions. They are not directly applicable to the web of Linked Data, where results are not bounded by a single source but might encompass several Linked Data sources. As opposed to the source selection problem [6], [7], which is focusing on computing the *most relevant sources*, the problem here is to compute the *most relevant combinations of sources*. The goal is to produce *routing plans*, which can be used to compute results from multiple sources. To this end, we provide the following contributions:

- We propose to investigate the problem of *keyword query routing* for keyword search over a large number of structured and Linked Data sources. Routing keywords only to relevant sources can reduce the high cost of searching for structured results that span multiple sources. To the best of our knowledge, the work presented in this paper represents the first attempt to address this problem.
- Existing work uses keyword relationships (KR) collected individually for single databases [6], [7]. We represent relationships between keywords as well as those between data elements. They are constructed for the entire collection of linked sources, and then grouped as elements of a compact *summary* called the *set-level keyword-element relationship graph* (KERG). Summarizing relationships is essential for addressing the scalability requirement of the Linked Data web scenario.
- IR-style ranking has been proposed to incorporate relevance at the level of keywords [7]. To cope with the increased keyword ambiguity in the web setting, we employ a *multilevel relevance model*, where elements to be considered are keywords, entities mentioning these keywords, corresponding sets of entities, relationships between elements of the same level, and inter-relationships between elements of different levels.
- We implemented the approach and evaluated it in a real-world setting using more than 150 publicly available data sets. The results show the applicability of this approach: valid plans (precision@1 = 0.92) that are highly relevant to the user information need (mean reciprocal rank (RR) = 0.86) can be computed in 1 second on average using a commodity PC. Further, we show that when routing is applied to an existing keyword search system to prune sources, substantial performance gain can be achieved.

• *The authors are with Institute AIFB, Karlsruhe Institute of Technology, 76128 Karlsruhe, Germany. E-mail: {thanh.tran, l.zhang}@kit.edu.*

Manuscript received 26 Jan. 2012; revised 14 Aug. 2012; accepted 10 Dec. 2012; published online 14 Jan. 2013.
Recommended for acceptance by S. Sudarshan.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2012-01-0065.
Digital Object Identifier no. 10.1109/TKDE.2013.13.

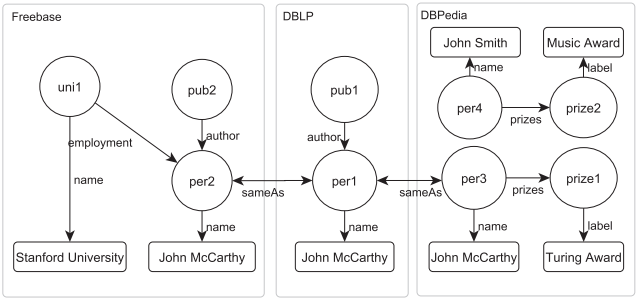


Fig. 1. Extract of the web data graph.

Outline. Section 2 provides an overview of existing work. The problem and solution are sketched in Section 3. The summary model is presented in Section 4 while Section 5 shows how it can be used to compute routing plans and Section 7 discusses how to rank them. Evaluation results are provided in Section 8 before we conclude in Section 9.

2 Related Work

There are two directions of work: 1) keyword search approaches compute the most relevant structured results and 2) solutions for source selection compute the most relevant sources.

2.1 Keyword Search

Existing work can be categorized into two main categories:

There are *schema-based approaches* implemented on top of off-the-shelf databases [8], [1], [2], [3], [9], [10]. A keyword query is processed by mapping keywords to elements of the database (called *keyword elements*). Then, using the schema, valid join sequences are derived, which are then employed to join ("connect") the computed keyword elements to form so-called candidate networks representing possible results to the keyword query.

Schema-agnostic approaches [11], [12], [13], [5] operate directly on the data. Structured results are computed by exploring the underlying data graph. The goal is to find structures in the data called Steiner trees (*Steiner graphs* in general), which connect keyword elements [13]. For the query "Stanford John Award" for instance, a Steiner graph is the path between $uni1$ and $prize1$ in Fig. 1. Various kinds of algorithms have been proposed for the efficient exploration of keyword search results over data graphs, which might be very large. Examples are bidirectional search [11] and dynamic programming [5].

Recently, a system called Kite extends schema-based techniques to find candidate networks in the multisource setting [4]. It employs schema matching techniques to discover links between sources and uses structure discovery techniques to find foreign-key joins across sources. Also based on precomputed links, Hermes [14] translates keywords to structured queries. However, experiments have been performed only for a small number of sources so far. Kite explicitly considered only the setting where "the number of databases that can be dealt with is up to the tens" [4].

In our scenario, the search space drastically increases, and also, the number of potential results may increase exponentially with the number of sources and links between them. Yet, most of the results may be not necessary especially when they are not relevant to the user. A solution to keyword query routing can address these problems by pruning unpromising sources and enabling users to select combinations that more likely contain relevant results. For the routing problem, we do not need to compute results capturing specific elements at the data level, but can focus on the more coarse-grained level of sources.

2.2 Database Selection

More closely related to this work are existing solutions to database selection, where the goal is to identify the most relevant databases. The main idea is based on modeling databases using keyword relationships. A keyword relationship is a pair of keywords that can be connected via a sequence of join operations. For instance, $\langle Stanford, Award\rangle$ is a keyword relationship as there is a path between $uni1$ and $prize1$ in Fig. 1. A database is relevant if its keyword relationship model covers all pairs of query keywords. M-KS [6] captures relationships using a matrix. Since M-KS considers only binary relationships between keywords, it incurs a large number of false positives for queries with more than two keywords. This is the case when all query keywords are pairwise related but there is no combined join sequence which connects all of them.

G-KS [7] addresses this problem by considering more complex relationships between keywords using a keyword relationship graph (KRG). Each node in the graph corresponds to a keyword. Each edge between two nodes corresponding to the keywords $\langle k_i, k_j\rangle$ indicates that there exists at least two connected tuples $t_i \leftrightarrow t_j$ that match k_i and k_j. Moreover, the distance between t_i and t_j are marked on the edges.

Compared to M-KS, G-KS computes more relevant sources due to these differences: G-KS adopts IR-style ranking to compute TF-IDF for keywords and for keyword relationships. Further, it helps to reduce the number of false positives. It provides an additional level of filtering, validating connections between keywords based on complex relationships and distance information in the KRG.

Both M-KS and G-KS assume that sources are independent and answers reside within one single source. The KRG, its keywords, relationships between keywords as well as scores are derived from and built for one single database. The solution we propose for modeling and scoring relationships is geared toward the entire Linked Data collection. Moreover, we make use of a summary, which instead of capturing relationships at the level of keywords (and data elements), it operates at the level of element sets.

3 Overview

In this section, we discuss the data, define the problem, and then briefly sketch the proposed solution.

3.1 Web of Data

We use a graph-based data model to characterize individual data sources. In that model, we distinguish between an *element-level data graph* representing relationships between

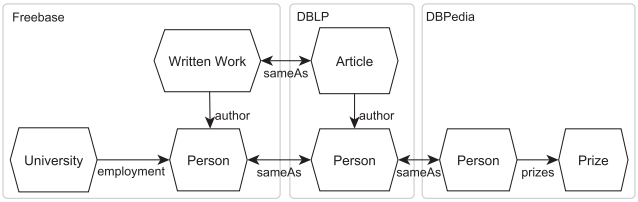


Fig. 2. Set-level web data graph.

individual data elements, and a *set-level data graph*, which captures information about group of elements.

Definition 1 (Element-level Data Graph). *An element-level data graph $g(\mathcal{N}, \mathcal{E})$ consists of*

- *the set of nodes $\mathcal{N}$, which is the disjoint union of $\mathcal{N}_\mathcal{E} \uplus \mathcal{N}_\mathcal{V}$, where the nodes $\mathcal{N}_\mathcal{E}$ represent entities and the nodes $\mathcal{N}_\mathcal{V}$ capture entities' attribute values, and*
- *the set of edges $\mathcal{E}$, subdivided by $\mathcal{E} = \mathcal{E}_\mathcal{R} \uplus \mathcal{E}_\mathcal{A}$, where $\mathcal{E}_\mathcal{R}$ represents interentity relations, $\mathcal{E}_\mathcal{A}$ stands for entity-attribute assignments. We have $e(n_1, n_2) \in \mathcal{E}_\mathcal{R}$ iff $n_1, n_2 \in \mathcal{N}_\mathcal{E}$ and $e(n_1, n_2) \in \mathcal{E}_\mathcal{A}$ iff $n_1 \in \mathcal{N}_\mathcal{E}$ and $n_2 \in \mathcal{N}_\mathcal{V}$.*

The set of attribute edges $\mathcal{E}_\mathcal{A}(n) = \{e(n, m) \in \mathcal{E}_\mathcal{A}\}$ is referred to as the description *of the entity n.*

Note that this model resembles RDF data where entities stand for some RDF resources, data values stand for RDF literals, and relations and attributes correspond to RDF triples. While it is primarily used to model RDF Linked Data on the web, such a graph model is sufficiently general to capture XML and relational data. For instance, a tuple in a relational database can be modeled as an entity, and foreign key relationships can be represented as interentity relations.

Definition 2 (Set-level Data Graph). *A* set-level data graph *of an element-level graph $g(\mathcal{N}_\mathcal{E} \uplus \mathcal{N}_\mathcal{V}, \mathcal{E}_\mathcal{R} \uplus \mathcal{E}_\mathcal{A})$ is a tuple $g' = (\mathcal{N}', \mathcal{E}')$. Every node $n' \in \mathcal{N}'$ stands for a set of element-level entities $\mathcal{N}_{n'} \subseteq \mathcal{N}_\mathcal{E}$, i.e., there is mapping* $\text{type} : \mathcal{N}_\mathcal{E} \mapsto \mathcal{N}'$ *that associates every element-level entity $n \in \mathcal{N}_\mathcal{E}$ with a set-level element $n' \in \mathcal{N}'$. Every edge $e'(n'_i, n'_j) \in \mathcal{E}'$ represents a relation between the two sets of element-level entities n'_i and n'_j. We have $\mathcal{E}' = \{e'(n'_i, n'_j) | e(n_i, n_j) \in \mathcal{E}_\mathcal{R}, \text{type}(n_i) = n'_i, \text{type}(n_j) = n'_j\}$.*

This set-level graph essentially captures a part of the Linked Data schema on the web that are represented in RDFS, i.e., relations between classes. Often, a schema might be incomplete or simply does not exist for RDF data on the web. In such a case, a pseudoschema can be obtained by computing a structural summary such as a dataguide [15]. A set-level data graph can be derived from a given schema or a generated pseudoschema. Thus, we assume a membership mapping $type : \mathcal{N}_\mathcal{E} \mapsto \mathcal{N}'$ exists and use $n \in n'$ to denote that n belongs to the set n'. An example of the set-level graph is given in Fig. 2.

We consider the search space as a set of Linked Data sources, forming a web of data.

Definition 3 (Web Graph). *The web of data is modeled as a* web graph *$\mathcal{W}^*(\mathcal{G}^*, \mathcal{N}^*, \mathcal{E}^*_i \uplus \mathcal{E}^*_e)$, where $\mathcal{G}^*$ is the set of all* data graphs, *$\mathcal{N}^*$ is the set of all nodes, $\mathcal{E}^*_i$ is the set of all "internal" edges that connect elements within a particular source, and $\mathcal{E}^*_e$ is the set of all "external" edges, which establish links between elements of two different sources, i.e., $\mathcal{G}^* = \{g_1(\mathcal{N}^*_1, \mathcal{E}^*_1), g_2(\mathcal{N}^*_2, \mathcal{E}^*_2), \ldots, g_n(\mathcal{N}^*_n, \mathcal{E}^*_n)\}, \mathcal{N}^* = \bigcup_{l=1}^{n} \mathcal{N}^*_l$, $\mathcal{E}^*_e = \{e(n_i, n_j) | n_i \in \mathcal{N}^*_i, n_j \in \mathcal{N}^*_j, \mathcal{N}^*_i \neq \mathcal{N}^*_j\}$, and $\mathcal{E}^* = \bigcup_{l=1}^{n} \mathcal{E}^*_{i_l} \cup \mathcal{E}^*_{e_l}$. When considering the nodes and edges only, we simply use $\mathcal{W}^*(\mathcal{N}^*, \mathcal{E}^*)$. We use $\mathcal{W}(\mathcal{G}, \mathcal{N}, \mathcal{E})$ to distinguish the* element-level web graph *from the* set-level web graph *$\mathcal{W}'(\mathcal{G}', \mathcal{N}', \mathcal{E}')$.*

TABLE 1
Notation

Symbols	Description
$k, \mathcal{K}$	keyword (term), keyword query
$\mathcal{N}, \mathcal{N}_\mathcal{E}, \mathcal{N}_\mathcal{V}$	graph, entity and attribute value nodes
$\mathcal{E}, \mathcal{E}_\mathcal{R}, \mathcal{E}_\mathcal{A}$	graph, relation and attribute edges
$\mathcal{E}_\mathcal{A}(n)$	description of entity n
$g, \mathcal{G}$	graph, set of graphs
$\mathcal{W}^*$; $\mathcal{W}, \mathcal{W}'$	Web graph; element-level and set-level Web graph
$\mathcal{W}'_\mathcal{K}$, $n'_\mathcal{K}$, $e'_\mathcal{K}$	set-level keyword-element relationship graph, node and edge
$\mathcal{W}^\mathcal{S}$	Steiner graph (keyword query result)
$\mathcal{W}'^\mathcal{S}_\mathcal{K}$	routing graph
$\mathcal{RP}$	routing plan
$\mathcal{M}^\mathcal{E}_g(d)$	matrix capturing paths in g of length d
$\mathcal{M}^{\mathcal{E}_e}_{(g_i, g_j)}(d)$	matrix capturing paths between g_i and g_j of length d
$tf(k_i, n'_\mathcal{K})$	frequency (count) of k_i in $n'_\mathcal{K}$

3.2 Keyword Query Routing

We aim to identify data sources that contain results to a keyword query. In the Linked Data scenario, results might combine data from several sources:

Definition 4 (Keyword Query Result). *A web graph $\mathcal{W}(\mathcal{N}, \mathcal{E})$ contains a result for a query $\mathcal{K} = \{k_1, k_2, \ldots, k_{|\mathcal{K}|}\}$ if there is a subgraph also called* Steiner graph *$\mathcal{W}^\mathcal{S}(\mathcal{N}^\mathcal{S}, \mathcal{E}^\mathcal{S})$, which for all $k_i \in \mathcal{K}$, contains a* keyword element *node $n_i \in \mathcal{N}^\mathcal{S}$ whose description $\mathcal{E}_\mathcal{A}(n_i)$ matches k_i, and there is a path between n_i and n_j ($n_i \leftrightarrow n_j$) for all $n_i, n_j \in \mathcal{N}^\mathcal{S}$.*

Typical for all keyword search approaches is the pragmatic assumption that users are only interested in compact results such that a threshold d_{max} can be used to constrain the connections to be considered. The type of Steiner graphs that is of particular interest is *d_{max}-Steiner graphs* $\mathcal{W}^\mathcal{S}(\mathcal{N}^\mathcal{S}, \mathcal{E}^\mathcal{S})$, where for all $n_i, n_j \in \mathcal{N}^\mathcal{S}$, paths between n_i and n_j is of length d_{max} or less. This work also relies on this assumption to constrain the size of the search space.

Definition 5 (Keyword Routing Plan). *Given the web graph $\mathcal{W} = (\mathcal{G}, \mathcal{N}, \mathcal{E})$ and a keyword query $\mathcal{K}$, the mapping $\mu : \mathcal{K} \rightarrow 2^\mathcal{G}$ that associates a query with a set of data graphs is called a* keyword routing plan *$\mathcal{RP}$. A plan $\mathcal{RP}$ is considered* valid *w.r.t. $\mathcal{K}$ when the union set of its data graphs contains a result for $\mathcal{K}$.*

The problem of keyword query routing is to find the top-k keyword routing plans based on their relevance to a query. A *relevant plan* should correspond to the information need as intended by the user.

Table 1 provides an overview of all symbols.

3.3 Multilevel Inter-Relationship Graph

We illustrate the search space of keyword query routing using a *multilevel inter-relationship graph*. At the lowest level,

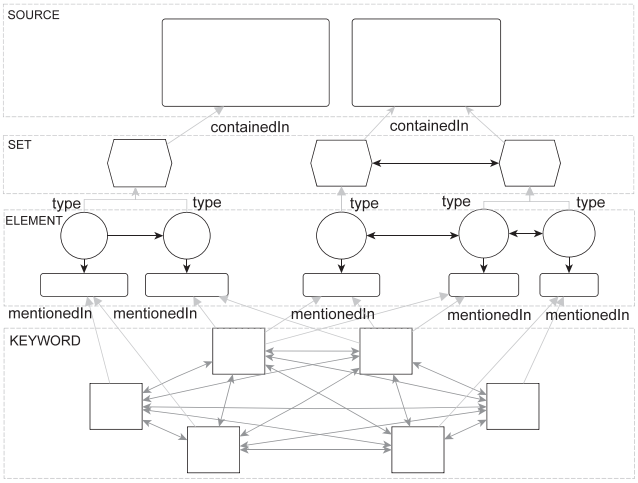


Fig. 3. Multilevel Inter-relationship graph.

it models relationships between keywords just like a KRG in the G-KS approach [7]. In the upper levels, there are $\mathcal{W}(\mathcal{N},\mathcal{E})$ followed by $\mathcal{W}'(\mathcal{N}',\mathcal{E}')$ and the *source-level web graph*, which contains sources as nodes.

The inter-relationships between elements at different levels are illustrated in Fig. 3. A keyword is *mentioned in* some entity descriptions at the element level. Entities at the element level are associated with a set-level element via *type*. A set-level element *is contained* in a source. There is an edge between two keywords iff two elements at the element level mentioning these keywords are connected via a path.

Fig. 3 represents a holistic view of the search space. Based on this view, we propose a ranking scheme that deals with relevance at many levels. Further, Fig. 3 provides different perspectives on the search space. It makes clear that there are different views on the data representing models and summaries at different levels of granularity. Based on this representation of the search space, we now discuss how existing work on keyword search and database selection can be extended to solve the problem of keyword query routing.

3.4 Basic Approaches

Existing work on keyword search [11], [12], [13] relies on an element-level model (i.e., data graphs) to compute keyword query results. Elements mentioning keywords are retrieved from this model and paths between them are explored to compute Steiner graphs. To deal with the keyword routing problem, elements can be stored along with the sources they are contained in so that this information can be retrieved to derive the routing plans from the computed keyword query results. Thus, existing keyword search solutions naturally apply to this problem. However, the data graph and the number of keyword elements are possibly very large in our scenario, and thus, exploring all paths between them in the data graphs is expensive.

KRG [7] captures relationships at the keyword level. As opposed to keyword search solutions, relationships captured by a KRG are not direct edges between tuples but stand for paths between keywords. For database selection, KRG relationships are retrieved for all pairs of query keywords to construct a subgraph. Based on these keyword relationships alone, it is not possible to guarantee that such a subgraph is also a Steiner graph (i.e., to guarantee that the database is relevant). To address this, subgraphs are validated by finding those that contain Steiner graphs. This is a filtering step, which makes use of information in the KRG as well as additional information about which keywords are contained in which tuples in the database. It is similar to the exploration of Steiner graph in keyword search, where the goal is to ensure that not only keywords but also tuples mentioning them are connected. Thus, this approach also relies on information at the element level. However, since KRG focuses on database selection, it only needs to know whether two keywords are connected by some join sequences or not. This information is stored as relationships in the KRG and can be retrieved directly. For keyword search, paths between data elements have to be retrieved and explored. Retrieving and exploring paths that might be composed of several edges are clearly more expensive than retrieving relationships between keywords (that capture "some" paths between elements mentioning these keywords, while the actual paths are not needed).

For extending this work to query routing, we need to construct a "multisource KRG" for the entire collection of sources. Such a KRG models both relationships that are within and those that link between sources. Keyword relationships are stored together with the elements they contained in, and are associated with source information. In particular, we construct an *element-level keyword-element relationship graph* (E-KERG) where constituents are relationships $e_{\mathcal{K}} = \langle n_{\mathcal{K}_i}, n_{\mathcal{K}_j} \rangle$ and every element $n_{\mathcal{K}} = (k, n, g)$ captures besides the graph g, both the keyword k and the element n in which it was mentioned (thus, we use the term "keyword-element"). Using E-KERG, one can retrieve connections between k_i and k_j, validate whether the corresponding entities n_i and n_j are connected, and finally extract source information to construct keyword routing plans.

While extending the KRG like this can solve the keyword query routing problem, it has one essential flaw. Just like keyword search approaches, the employed model captures relationships at the level of elements. Since a model has to be constructed for the entire collection of sources, the number of relationships might be very large. For reducing the complexity of the search space, we will now propose a more compact model.

4 Set-Level Summary

The summary we derive is not at the level of elements but set of elements.

4.1 Set-Level KERG

Definition 6 (Set-level KERG). *Given* $\mathcal{W}(\mathcal{G},\mathcal{N},\mathcal{E})$ *and* $\mathcal{W}'(\mathcal{G}', \mathcal{N}', \mathcal{E}')$, *a* set-level keyword-element relationship graph *is a tuple* $\mathcal{W}'_{\mathcal{K}} = (\mathcal{N}'_{\mathcal{K}}, \mathcal{E}'_{\mathcal{K}})$. *Every keyword-element node* $n'_{\mathcal{K}} \in \mathcal{N}'_{\mathcal{K}}$ *is a tuple* (k, n', g) *where* $n' \in \mathcal{N}'$ *is the corresponding set-level node it represents,* $g \in \mathcal{G}$ *is the graph it is associated with, and* k *is a keyword that matches some descriptions* $\mathcal{E}_{\mathcal{A}}(n)$ *of the entities* $n \in n'$. *There is a* keyword-element relationship *(KER)* $e'_{\mathcal{K}} = \langle n'_{\mathcal{K}_i}, n'_{\mathcal{K}_j} \rangle \in \mathcal{E}'_{\mathcal{K}}$ *representing a*

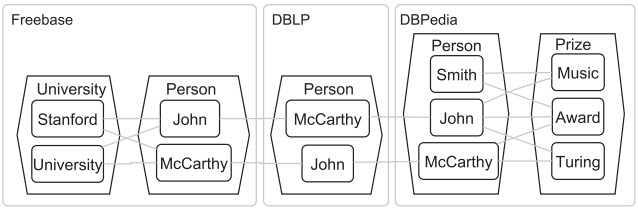


Fig. 4. A Set-level 1-KERG.

relationship between the keyword-element $n'_{\mathcal{K}_i} = (k_i, n'_i, g_i)$ *and* $n'_{\mathcal{K}_j} = (k_j, n'_j, g_j)$*, iff there is an* $n_i \in n'_i$ *that mentions* k_i*, an* $n_j \in n'_j$ *that mentions* k_j*, and* $n_i \leftrightarrow n_j$*. A* d_{max}*-KERG is a KERG, which captures only paths between* n_i *and* n_j *with length* d_{max} *or less* ($n_i \leftrightarrow^{d_{max}} n_j$).

Intuitively, a d_{max}-KERG represents all paths between keywords that are connected over a maximum distance d_{max}. This is to capture all d_{max}-Steiner graphs that exist in the data. However, it does not capture paths exhibited by data elements but paths between set of elements. As discussed in the following example, certain relationships at the level of elements are combined and summarized to one set-level relationship.

Example 1. A KERG for our running example with $d_{max} = 1$ is illustrated in Fig. 4. For instance, there is a keyword-element node $(John, Person, DBPedia)$. Note that the relationship $\langle (John, Person, DBPedia), (Award, Prize, DBPedia)\rangle$ actually stands for the element-level connections $\langle (John, per3, DBPedia), (Award, prize1, DBPedia)\rangle$, and $\langle (John, per4, DBPedia), (Award,\ prize2, DBPedia)\rangle$ because $per3$ and $per4$ mention $John, prize1$ and $prize2$ mention $Award$, $per3$, $per4 \in Person$, $prize1$, $prize2 \in Prize$, there is a path between $per3$ and $prize1$, and a path between $per4$ and $prize2$ (see web data graph in Fig. 1). This example illustrates that element-level relationships, which share the same pair of terms ($John$ and $Award$), classes ($Person$ and $Prize$), and sources ($DBPedia$ and $DBPedia$) can be summarized to one single set-level relationship.

For constructing KERG, we extract keywords and relationships from the data. Then, based on the elements and sets of elements in which they occur, we create keyword-element relationships. Precomputing relationships (i.e., paths) between data elements are typically performed for keyword search to improve online performance. These relationships are stored in specialized indexes and retrieved at the time of keyword query processing to accelerate the search for Steiner graphs [12]. For database selection, relationships between keywords are also precomputed [6]. This work neither considers relationships between keywords nor relationships between data elements but between keyword-elements that collectively represent the keywords and the data elements in which they occur. They are represented as keyword-element relationships. Further, different from previous works [6], [7], not only relationships within a particular source but also relationships between sources are taken into account. We now discuss the methods for computing these relationships and for constructing KERG.

Relationships within Data Sources. For computing relationships within individual sources, we model paths between nodes in a source $g(\mathcal{N}_\mathcal{E} \uplus \mathcal{N}_\mathcal{V}, \mathcal{E})$ as $|\mathcal{N}_\mathcal{E}| \times |\mathcal{N}_\mathcal{E}|$ symmetric Boolean matrices $\mathcal{M}^\mathcal{E}_g(d)$ w.r.t. various distances d. $\mathcal{M}^\mathcal{E}_g(1)$, illustrated as

$$\mathcal{M}^\mathcal{E}_g(1) = \left\{ \begin{array}{ccccc} & n_1 & n_2 & \cdots & n_n \\ n_1 & 0 & 1 & \cdots & 1 \\ n_2 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & & & \\ n_n & 1 & 0 & \cdots & 0 \end{array} \right\},$$

represents direct paths between all entity pairs. An entry r^d_{ij} in this matrix is 1, iff there is a path between n_i and n_j of length d, where n_i and n_j denote the entities at ith row and jth column in $\mathcal{M}^\mathcal{E}_g(d)$; otherwise, r^d_{ij} is 0. For $1 < d \leq d_{max}$, the matrix $\mathcal{M}^\mathcal{E}_g(d)$ is constructed iteratively using the formula $\mathcal{M}^\mathcal{E}_g(d) = (r^d_{ij})_{n\times n} = \mathcal{M}^\mathcal{E}_g(d-1) \times \mathcal{M}^\mathcal{E}_g(1)$, which represents paths between entities with length no greater than d.

Relationships between Data Sources. For computing relationships between sources of the web graph $\mathcal{W}(\mathcal{G}, \mathcal{N}, \mathcal{E}_i, \mathcal{E}_e)$, we model external links $e \in \mathcal{E}_e$ between every data source pair $\langle g_i(\mathcal{N}_{\mathcal{E}_i} \uplus \mathcal{N}_{\mathcal{V}_i}, \mathcal{E}_i), g_j(\mathcal{N}_{\mathcal{E}_j} \uplus \mathcal{N}_{\mathcal{V}_j}, \mathcal{E}_j)\rangle$ as a $|\mathcal{N}_{\mathcal{E}_i}| \times |\mathcal{N}_{\mathcal{E}_j}|$ Boolean matrix $\mathcal{M}^{\mathcal{E}_e}_{(g_i,g_j)}$. In $\mathcal{M}^{\mathcal{E}_e}_{(g_i,g_j)}$, an entry is 1 iff $e(n_i, n_j) \in \mathcal{E}_e$, where n_i and n_j denote the entities at ith row and jth column in $\mathcal{M}^{\mathcal{E}_e}_{(g_i,g_j)}$; otherwise, it is 0. Then, we compute all relationships using the following matrix:

Theorem 1. *All paths between nodes of the graph* g_i *and nodes of the graph* g_j *with maximum length* d *are captured by a* $|\mathcal{N}_{\mathcal{E}_i}| \times |\mathcal{N}_{\mathcal{E}_j}|$ *Boolean matrix* $\mathcal{M}^\mathcal{E}_{(g_i,g_j)}(d)$*. We have* $\mathcal{M}^\mathcal{E}_{(g_i,g_j)}(0) = 1$ *and* $\mathcal{M}^\mathcal{E}_{(g_i,g_j)}(1) = \mathcal{M}^{\mathcal{E}_e}_{(g_i,g_j)}$*. For* $d > 1$*,*

$$\mathcal{M}^\mathcal{E}_{(g_i,g_j)}(d) = \sum_{(d_i,d_j)\in\mathcal{P}(\mathbb{N}_0), d_i+1+d_j\leq d} \mathcal{M}^\mathcal{E}_{g_i}(d_i) \times \mathcal{M}^{\mathcal{E}_e}_{(g_i,g_j)} \times \mathcal{M}^\mathcal{E}_{g_j}(d_j), \quad (1)$$

where $\mathcal{P}(\mathbb{N}_0)$ *denotes 2-elements permutations of the set* $\mathbb{N}_0$*.*

Proof Sketch. Every path $n_{g_i} \leftrightarrow n_{g_j}$ consists of three parts, namely the subpath $p_i = n_{g_i} \leftrightarrow n^*_{g_i}$ that is completely covered by g_i, the subpath $p_j = n^*_{g_j} \leftrightarrow n_{g_j}$ covered by g_j, and the external edge $p_{ij} = n^*_{g_i} \leftrightarrow n^*_{g_j}$, which connects p_i with p_j. The paths p_{ij} represent the shortest relationships between elements of g_i and g_j with $d = 1$. All paths between elements of g_i and g_j have distance $d = d_i + 1 + d_j$, where d_i and d_j denote the length of p_i and p_j, respectively. The matrix multiplication $\mathcal{M}^\mathcal{E}_{g_i}(d_i) \times \mathcal{M}^{\mathcal{E}_e}_{(g_i,g_j)} \times \mathcal{M}^\mathcal{E}_{g_j}(d_j)$ gives all the paths with distance $d = d_i + 1 + d_j$. The matrix addition over all possible permutations of (d_i, d_j), where $d_i + 1 + d_j \leq d$ gives all the paths, which have distance between 1 and d. □

Constructing Keyword-Element Relationships. First, all terms are extracted from the data. For each pair of terms $\langle k_i, k_j\rangle$, if there exist two nodes n_i and n_j where $k_i \in \mathcal{E}_\mathcal{A}(n_i)$ and $k_j \in \mathcal{E}_\mathcal{A}(n_j)$ and $n_i \leftrightarrow n_j$, or when k_i and k_j are mentioned in the description of the same element ($n_i = n_j$), we create an element-level KER $\langle n_{\mathcal{K}_i}, n_{\mathcal{K}_j}\rangle$. Two element-level KERs $\langle n_{\mathcal{K}_{i_1}}(k_i, n_{i_1}, g_i), n_{\mathcal{K}_{j_1}}(k_j, n_{j_1}, g_j)\rangle$ and $\langle n_{\mathcal{K}_{i_2}}(k_i, n_{i_2}, g_i), n_{\mathcal{K}_{j_2}}(k_j, n_{j_2}, g_j)\rangle$ are grouped to one single

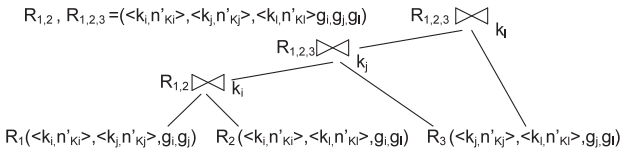


Fig. 5. Join plan for three keywords.

set-level KER $\langle n'_{\mathcal{K}_i}(k_i, n'_i, g_i), n'_{\mathcal{K}_j}(k_j, n'_j, g_j)\rangle$ iff $n_{i_1}, n_{i_2} \in n'_i$ and $n_{j_1}, n_{j_1} \in n'_j$. For computing all set-level KERs, this summarization is iteratively performed for all possible pairs of element-level KERs.

5 Computing Routing Plans

Routing plans are computed by searching for Steiner graphs in the summary. Since edges in the summary stand for paths between elements, subgraphs of the summary capturing Steiner graphs can be defined as follows:

Definition 7 (Routing Graph). *Given a keyword query $\mathcal{K} = \{k_1, k_2, \ldots, k_{|\mathcal{K}|}\}$ and a KERG $\mathcal{W}'_{\mathcal{K}}(\mathcal{N}'_{\mathcal{K}}, \mathcal{E}'_{\mathcal{K}})$, a routing graph $\mathcal{W}'^{\mathcal{S}}_{\mathcal{K}}(\mathcal{N}'^{\mathcal{S}}_{\mathcal{K}}, \mathcal{E}'^{\mathcal{S}}_{\mathcal{K}})$ is a subgraph of $\mathcal{W}'_{\mathcal{K}}$ that satisfies the following properties:*

- *For each query keyword $k \in \mathcal{K}$, there is a node $n'_{\mathcal{K}_i}(k, n', g) \in \mathcal{N}'^{\mathcal{S}}_{\mathcal{K}}$.*
- *It is a complete graph, i.e., for every pair of nodes $n'_{\mathcal{K}_i}, n'_{\mathcal{K}_j} \in \mathcal{N}'^{\mathcal{S}}_{\mathcal{K}}$ there is an edge $\langle n'_{\mathcal{K}_i}, n'_{\mathcal{K}_j}\rangle \in \mathcal{E}'^{\mathcal{S}}_{\mathcal{K}}$.*

Thus, we call Steiner graphs that can be found in the summary routing graphs. Similar to a routing plan, a routing graph contains a set of data sources. Moreover, a routing graph contains information that enables the user to assess whether it is relevant: a plan is relevant only if the nodes mentioning the keywords and relationships between them correspond to the intended information need. This additional information will be used in the evaluation to assess the effectiveness of ranking.

As discussed, unlike keyword search, which requires expensive exploration of edges between elements (at the element-level), relationships between keywords capturing possibly complex paths can be directly retrieved and employed for routing. Basically, the computation can be divided into three stages: 1) computation of routing graphs, 2) aggregation of routing graphs, and 3) ranking query routing plans. First, for each query keyword pair, corresponding relationships are retrieved from KERG. They are joined according to a plan (as illustrated for three keywords k_i, k_j, k_l in Fig. 5, they act as join variables) to obtain routing graphs. Note in Fig. 5 that for obtaining complete graphs, it is necessary to join on all three keywords. While other standard join implementations can be used, we employ hash join in our experiment. Because the order of joins does not have an effect on the top-k procedure, we employ in the end (discussed in Section 7), pairs of keywords are chosen randomly and iteratively until all query keywords are covered.

The procedure for computing routing plans is described in Algorithm 1. Given a query $\mathcal{K}$ and the summary $\mathcal{W}'_{\mathcal{K}}$, the algorithm computes a set of routing plans $[\mathcal{RP}]$. For this, it first determines the join plan $\mathcal{JP}$. Based on this plan, KERG relationships are retrieved for every keyword pair, and joined with the intermediate result table $\mathcal{T}$. This table contains candidate routing graphs, including the scores of their constituent elements and their combined score. When the join plan is worked off, the combined score is computed for every tuple in $\mathcal{T}$, i.e., for every routing graph $\mathcal{W}'^{\mathcal{S}}_{\mathcal{K}}$. Routing graphs, which represent the same set of sources, are aggregated into one single result. This is because we want to output only those plans that capture unique combination of sources. The score of a plan is computed from the scores of its routing graphs according to a ranking function that we will discuss in the next section.

Algorithm 1: PPRJ: ComputeRoutingPlan($\mathcal{K}, \mathcal{W}'_{\mathcal{K}}$)

Input: The query $\mathcal{K}$, the summary $\mathcal{W}'_{\mathcal{K}}(\mathcal{N}'_{\mathcal{K}}, \mathcal{E}'_{\mathcal{K}})$
Output: Set of routing plans $[\mathcal{RP}]$
$\mathcal{JP} \leftarrow$ a join plan that contains all $\langle k_i, k_j\rangle \in 2^{\mathcal{K}}$;
$\mathcal{T} \leftarrow$ a table where every tuple captures a join sequence of KERG relationships $e'_{\mathcal{K}} \in \mathcal{E}'_{\mathcal{K}}$, the score of each $e'_{\mathcal{K}}$, and the combined score of the join sequence; it is initially empty;
while $\neg\mathcal{JP}.empty()$ **do**
  $\langle k_i, k_j\rangle \leftarrow \mathcal{JP}$.pop() ;
  $\mathcal{E}'_{\langle k_i,k_j\rangle} \leftarrow$ retrieve($\mathcal{E}'_{\mathcal{K}}, \langle k_i, k_j\rangle$);
  if $\mathcal{T}.empty()$ **then**
    $\mathcal{T} \leftarrow \mathcal{E}'_{\langle k_i,k_j\rangle}$;
  else
    $\mathcal{T} \leftarrow \mathcal{E}'_{\langle k_i,k_j\rangle} \bowtie \mathcal{T}$;
Compute scores of tuples in $\mathcal{T}$ via SCORE($\mathcal{K}, \mathcal{W}'^{\mathcal{S}}_{\mathcal{K}}$);
$[\mathcal{RP}] \leftarrow$ Group $\mathcal{T}$ by sources to identify unique combinations of sources;
Compute scores of routing plans in $[\mathcal{RP}]$ via SCORE($\mathcal{K}, \mathcal{RP}$);
Sort $[\mathcal{RP}]$ by score;

Example 2. For the query $\mathcal{K} = \{Stanford, John, Award\}$, we retrieve the keyword-element relationships

$$\langle (Stanford, University, Freebase), (John, Person, Freebase)\rangle,$$
$$\langle (Stanford, University, Freebase), (John, Person, DBLP)\rangle,$$

and $\langle (Stanford, University, Freebase), (John, Person, DBPedia)\rangle$ for the pair of keywords $\langle Stanford, John\rangle$ and proceed in the same way for the other pairs $\langle John, Award\rangle$ and $\langle Stanford, Award\rangle$. We join these relationships in the order $\langle Stanford, John\rangle \bowtie_{John} \langle John, Award\rangle \bowtie_{Stanford} \langle Stanford, Award\rangle \bowtie_{Award} \langle Stanford, Award\rangle$. One resulting join sequence is: 1) first

$$\langle (Stanford, University, Freebase), (John, Person, DBPedia)\rangle \bowtie_{John} \langle (John, Person, DBPedia), (Award, Prize, DBPedia)\rangle$$

(i.e., join these two keyword-element relationships on the keyword-element retrieved for $John$, which is $(John, Person, DBPedia)$), 2) then join this intermediate

result with $\langle(Stanford, University, Freebase), (Award, Prize, DBPedia)\rangle$ on the element retrieved for $Stanford$, and 3) finally, join with

$$\langle(Stanford, University, Freebase), (Award, Prize, DBPedia)\rangle$$

on the element retrieved for *Award*. The result of this join sequence is the path

$$\langle(Stanford, University, Freebase), (John, Person, DBPedia), (Award, Prize, DBPedia)\rangle,$$

representing the one and only routing graph of this example. Thus, no aggregation needs to be performed and the only routing plan derived from this is $\{Freebase, DBPedia\}$.

6 Summary and Algorithm Analysis

In this section, we analyze completeness, soundness, and the complexity of using KERG.

6.1 Completeness

Given there are keyword query results, KERG can be used to compute all routing graphs:

Theorem 2. *Given $\mathcal{K}$ and the graph $\mathcal{W}$, there exists a routing graph $\mathcal{W}'^{\mathcal{S}}_{\mathcal{K}}$ if $\mathcal{W}$ contains a result for $\mathcal{K}$.*

Proof Sketch. A web graph $\mathcal{W}(\mathcal{N}, \mathcal{E})$ contains a result if there is a Steiner graph $\mathcal{W}^{\mathcal{S}}(\mathcal{N}^{\mathcal{S}}, \mathcal{E}^{\mathcal{S}})$ in $\mathcal{W}$ where for all query keywords k_i, there is a corresponding keyword element $n_i \in \mathcal{N}^{\mathcal{S}}$, and $n_i \leftrightarrow n_j$ for all $n_i, n_j \in \mathcal{N}^{\mathcal{S}}$. Recall that a routing graph is derived from the set-level KERG $\mathcal{W}'_{\mathcal{K}}(\mathcal{N}'_{\mathcal{K}}, \mathcal{E}'_{\mathcal{K}})$. By Definition 7, a routing graph is essentially a Steiner graph at the summary level. Thus, we have to show that $\mathcal{W}'_{\mathcal{K}}$ contains a Steiner graph if $\mathcal{W}$ does, i.e., for all keyword elements $n_i \in \mathcal{N}^{\mathcal{S}}$ and all pairs of connected elements $n_i, n_j \in \mathcal{N}^{\mathcal{S}}$ s.t. and $n_i \leftrightarrow n_j$, there are corresponding elements $n'_{\mathcal{K}_i}, n'_{\mathcal{K}_j} \in \mathcal{N}'_{\mathcal{K}}$, and $n'_{\mathcal{K}_i} \leftrightarrow n'_{\mathcal{K}_j}$. These correspondences follow from Definition 6 stating that for all pairs of keywords $\langle k_i, k_j\rangle$ iff there is an $n_i \in n'_i$ that mentions k_i, an $n_j \in n'_j$ that mentions k_j, and $n_i \leftrightarrow n_j$ then there are corresponding elements $n'_{\mathcal{K}_i} = (k_i, n'_i, g_i)$ and $n'_{\mathcal{K}_j} = (k_j, n'_j, g_j)$, and relationship $e'_{\mathcal{K}} = \langle n'_{\mathcal{K}_i}, n'_{\mathcal{K}_j}\rangle \in \mathcal{E}'_{\mathcal{K}}$. □

6.2 Soundness

While completeness is guaranteed, the use of summaries however sacrifices soundness, i.e., there are cases where the computed graphs are not necessarily valid:

Theorem 3. *The existence of a routing graph $\mathcal{W}'^{\mathcal{S}}_{\mathcal{K}}$ does not guarantee that there exists a result for $\mathcal{K}$.*

Proof Sketch. Given the query $\mathcal{K}$, a routing graph $\mathcal{W}'^{\mathcal{S}}_{\mathcal{K}}(\mathcal{N}'^{\mathcal{S}}_{\mathcal{K}}, \mathcal{E}'^{\mathcal{S}}_{\mathcal{K}})$ and $\mathcal{W}$, which represents the union set of sources captured by $\mathcal{W}'^{\mathcal{S}}_{\mathcal{K}}$, $\mathcal{W}'^{\mathcal{S}}_{\mathcal{K}}$ is valid if a Steiner graph $\mathcal{W}_{\mathcal{S}}$ can be constructed from $\mathcal{W}$. This is the case if $\mathcal{W}$ contains keyword elements for all $k_i \in \mathcal{K}$, and these elements are pairwise connected. We just showed that if keyword elements and paths between them exist at the element-level, then corresponding elements and relationships can also be found in $\mathcal{W}'$ and the summary $\mathcal{W}'_{\mathcal{K}}$, respectively. We now show that this is not true vice versa: for a set of three keywords k_i, k_j, k_l or more, the existence of the keyword-element relationships $\langle n'_{\mathcal{K}_i}, n'_{\mathcal{K}_j}\rangle$ and $\langle n'_{\mathcal{K}_j}, n'_{\mathcal{K}_l}\rangle$ in $\mathcal{W}'_{\mathcal{K}}$, where some elements captured by $n'_{\mathcal{K}_i}, n'_{\mathcal{K}_j}$, and $n'_{\mathcal{K}_l}$ mention k_i, k_j, and k_l, respectively (i.e., $n'_{\mathcal{K}_i}, n'_{\mathcal{K}_j}$, and $n'_{\mathcal{K}_l}$ are connected keyword elements at the summary level), does not imply the existence of the paths $\langle n_i, n_j\rangle$ and $\langle n_j, n_l\rangle$ in $\mathcal{W}$, where n_i, n_j, and n_l mention k_i, k_j, and k_l, respectively (i.e., n_i, n_j, n_l are connected keyword elements at the element level).

Assuming an element- and set-level graph g and g' like in our running example and

$$k_1 = John, k_2 = Stanford, k_3 = Music, k_1 \in \mathcal{E}_{\mathcal{A}}(per3), k_1 \in \mathcal{E}_{\mathcal{A}}(per4), k_2 \in \mathcal{E}_{\mathcal{A}}(uni1), k_3 \in \mathcal{E}_{\mathcal{A}}(prize2),$$

and $per3 \leftrightarrow uni1$, $per4 \leftrightarrow prize2$, and $per3$,$per4 \in Person$, $uni1 \in University$, $prize2 \in Prize$. Then, the KERG relationships derived from that include $\langle(John, Person, DBPedia), (Stanford, University, Freebase)\rangle$ and $\langle(John, Person, DBPedia), (Music, Prize, DBPedia)\rangle$. Thus, it can be derived from KERG that there exists a relationship between the elements representing the keywords $John$, $Stanford$, and $Music$ while in fact at the element level, the $John(per3)$ that is connected with $Stanford$ is not the same as the $John(per4)$ that is connected with $Music$. □

6.3 Complexity

For computing routing plans, elements retrieved from KERG have to be joined. In the worst case, a join operation requires $|input_i| \times |input_j|$ time and space. Inputs are joined for all possible pairs of keywords. This results in complexity $O(input_{max}^{C(\mathcal{K},2)})$, where $input_{max}$ here refers to the largest number of relationships that can be obtained for a keyword pair, $C(\mathcal{K}, 2)$ is the number of 2-element combinations of the set $\mathcal{K}$, and $C(\mathcal{K}, 2) - 1$ is the number of joins. However, in practice, inputs can be sorted such that efficient implementations like merge join can be applied. As a result, a join requires only $|input_i| + |input_j|$ such that overall complexity is $O(input_{max} \cdot C(\mathcal{K}, 2))$.

While the number of joins is same for all kinds of summary models (when using the same proposed procedure), the size of $input_{max}$ varies. Clearly, the more coarse grained the grouping, i.e., the higher the number of elements aggregated to one group at the summary level, the smaller will be $input_{max}$. Compared to E-KERG, KERG is potentially smaller because it summaries relationships at the element level (i.e., E-KERG relationships), which share the same pair of terms, classes, and sources. In the extreme case where every data element mentions only distinct terms, KERG and E-KERG are equal in size. In practice, elements in the same set have many terms in common such that KERG can yield substantial performance gain.

7 Ranking Routing Plans

To produce relevant plans, which yield results corresponding to the user information need, we propose a multilevel IR-style ranking scheme.

Keyword-element Score. Analogous to the notions of term frequency tf and inverse document frequency idf used in IR, we define the frequency $tf(k_i, n'_{\mathcal{K}})$ for a term k_i w.r.t. a keyword-element node $n'_{\mathcal{K}}(k, n', g)$ as the number of entities $n \in n'$, whose descriptions contain k_i, i.e.,

$$tf(k_i, n'_{\mathcal{K}}(k, n', g)) = |\{n \mid n \in n', k_i \in \mathcal{E}_{\mathcal{A}}(n))\}| \quad (2)$$

and

$$idf(k_i) = \ln \frac{|\mathcal{N}'_{\mathcal{K}}|}{|\{n'_{\mathcal{K}} \mid n'_{\mathcal{K}} \in \mathcal{N}'_{\mathcal{K}}, tf(k_i, n'_{\mathcal{K}}) > 0\}|}, \quad (3)$$

where the numerator denotes the number of keyword-element nodes in KERG and the denominator stands for the number of nodes $n'_{\mathcal{K}}$ that mention k_i. While $tf(k_i, n'_{\mathcal{K}})$ can be regarded as a measure for the descriptiveness of k_i w.r.t. the elements $n'_{\mathcal{K}}$, $idf(k_i)$ stands for the distinctiveness of k_i. Intuitively, a higher $tf(k_i, n'_{\mathcal{K}})$ means that more likely, $n'_{\mathcal{K}}$ is about k_i, while a higher $idf(k_i)$ reflects that k_i is a less common term. Just like in IR, we combine these two metrics to evaluate the relevance of a routing graph node $n'_{\mathcal{K}}$ w.r.t. k_i using

$$\mathcal{S}(k_i, n'_{\mathcal{K}}) = tf(k_i, n'_{\mathcal{K}}) * idf(k_i). \quad (4)$$

Keyword-element Relationship Score. In the same IR fashion, we propose metrics to measure the descriptiveness and distinctiveness of connected term pairs. Analogous to keyword-element scores, we define the frequency $tf(\langle k_i, k_j\rangle, e'_{\mathcal{K}}, d)$, which denotes the number of keyword-element relationships $e_{\mathcal{K}} = \langle n_{\mathcal{K}_i}(k_i, n'_i, g_i), n_{\mathcal{K}_j}(k_j, n'_j, g_j)\rangle$ at distance d, which have been summarized to one KERG relationship $e'_{\mathcal{K}} = \langle n'_{\mathcal{K}_i}, n'_{\mathcal{K}_j}\rangle$; and $idf(\langle k_i, k_j\rangle)$ is computed from the total number of KERG relationships and the ones where $tf(\langle k_i, k_j\rangle, e'_{\mathcal{K}}, d) > 0$. In this case, a higher $tf(\langle k_i, k_j\rangle, e'_{\mathcal{K}}, d)$ means that more likely, $e'_{\mathcal{K}}$ is about the keyword relationship $\langle k_i, k_j\rangle$, while a higher $idf(\langle k_i, k_j\rangle)$ means that the keyword relationship $\langle k_i, k_j\rangle$ is less frequent in the data. We combine them to obtain the score

$$\mathcal{S}(\langle k_i, k_j\rangle, e'_{\mathcal{K}}) = \sum_{0 \le i \le d} \varphi_d * tf(\langle k_i, k_j\rangle, e'_{\mathcal{K}}, d) * idf(\langle k_i, k_j\rangle), \quad (5)$$

where $\varphi_d = \frac{1}{d+1}$ is a monotonically decreasing function recognizing that when a path connecting two terms has a greater length, its structural relevance for these terms should be smaller.

Ranking Keyword Routing Plan. Incorporating the relevance score of constituent keyword-elements and keyword-element relationships, the ranking of a routing graph $\mathcal{W}^{\mathcal{S}}_{\mathcal{K}}(\mathcal{N}'^{\mathcal{S}}_{\mathcal{K}}, \mathcal{E}'^{\mathcal{S}}_{\mathcal{K}})$ w.r.t. a query $\mathcal{K}$ is computed as a combination of scores

$$\mathcal{S}(\mathcal{K}, \mathcal{W}'^{\mathcal{S}}_{\mathcal{K}}(\mathcal{N}'^{\mathcal{S}}_{\mathcal{K}}, \mathcal{E}'^{\mathcal{S}}_{\mathcal{K}})) = \sum_{e'_{\mathcal{K}}(n'_{\mathcal{K}_i}(k_i, n'_i, g_i), n'_{\mathcal{K}_j}(k_j, n'_j, g_j)) \in \mathcal{E}'^{\mathcal{S}}_{\mathcal{K}}} \mathcal{S}(\langle k_i, k_j\rangle, e'_{\mathcal{K}}) * (\mathcal{S}(k_i, n'_{\mathcal{K}_i}) + \mathcal{S}(k_j, n'_{\mathcal{K}_j})).$$

Finally, the rank of a keyword routing plan $\mathcal{RP}$ is computed as the aggregation

$$\mathcal{S}(\mathcal{K}, \mathcal{RP}) = \sum_{\mathcal{W}^{\mathcal{S}}_{\mathcal{K}} \in TOP\_K(\mathcal{RP})} \mathcal{S}(\mathcal{K}, \mathcal{W}^{\mathcal{S}}_{\mathcal{K}}), \quad (6)$$

where $TOP\_K(\mathcal{RP})$ returns the k-best ranked routing graphs in $\mathcal{RP}$. Using only k-best ones enables us to control the quality and also, not to associate a high rank to those plans, which can be derived from a large number but less relevant graphs.

While the proposed scheme also leverages TF-IDF like the ones used by M-KS and its extension G-KS, it addresses a different ranking problem, i.e., ranking combinations of sources instead of single sources. It does not consider every source independently. Node and relationship scores are derived from *statistics computed for the entire collection* of sources. Moreover, this scheme is more fine-granular, considering several levels of the search space. As opposed to G-KS, not only the scores of nodes and relationships at the keyword level but also their contributions to the score of keyword-element relationships and routing graphs at the set level are incorporated.

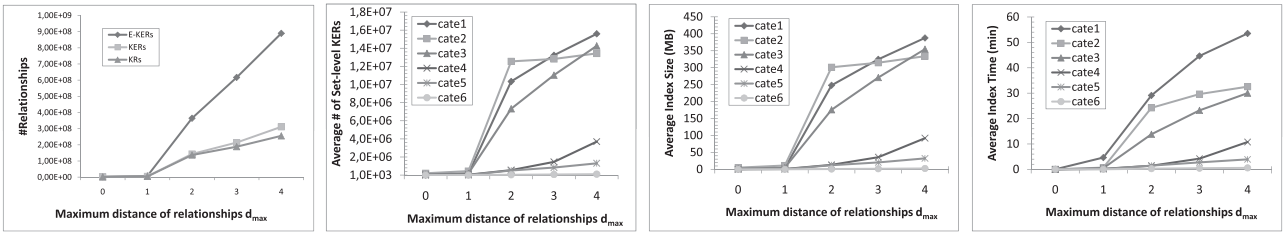
For estimating the score of a KRG node w.r.t. k, G-KS considers all tuples in the database that mention k. Thus, tuples have an effect on the final score even when they do not participate in the final answer. The same problem applies to KRG relationship scores. The score of a KERG element (relationship) w.r.t. k is computed not from all elements (tuples) mentioning k but only those that participate in the set-level element (relationship).

G-KS prunes invalid term relationships by considering the underlying paths and their distance to reduce false positives. However, this additional pruning step does not have an effect on the scores. This pruning step is similar to the computation of routing graph in our approach. In fact, pruning is a by-product of this computation because during this process, keyword-element relationships are successively retrieved and ruled out if they cannot contribute to the final routing graphs. As opposed to G-KS, scores of elements and relationships which have been pruned do not have an effect on the final score. Also, only the scores of some top routing graphs are incorporated into the overall score of the plan.

Top-k Plans. As demonstrated, we have reduced the problem of computing plans to the standard problem of processing join queries, for which there exist different kinds of optimization. Now, after the scores have been defined, we show that for instance, top-k algorithms such as rank-join [2] can be applied to output the k best results and to terminate early. Rank join takes a query plan with a fixed join order such as the one used in the previous section. In fact, the order of join operators can be chosen randomly as with rank join, the order of joining inputs (their entries) is not dependent on the order of operators but mainly determined by the pulling strategy, which basically prefers input entries that more likely lead to top-k results (those with high scores). Thus, rank join requires inputs that are sorted on their score, and assumes a monotonic score aggregation function for computing the score of the joined result. As a rank join input, we use

TABLE 2
Data Set Statistics

Category	#Sources	#Triples in Every Source	Average #Triples	Average #Entities	Average #Entity Sets	Example Source	#Triples in Ex. Source
Cate1	2	$1M \leq N$	1.6M	262,114	6	identi.ca	1.0M
Cate2	2	$500K \leq N < 1M$	715K	85,672	244	opencyc.org	613K
Cate3	8	$100K \leq N < 500K$	213K	50,542	55	dbtune.org	149K
Cate4	17	$50K \leq N < 100K$	74K	9,218	55	semanticweb.org	58K
Cate5	53	$10K \leq N < 50K$	21K	3,197	41	korrekt.org	21K
Cate6	72	$3K \leq N < 10K$	5K	745	9	eswc2006.org	3K



(a) #KR, #E-KERs, #KERs (b) #KERs (c) KERG size (d) KERG building time

Fig. 6. Element-level and Set-level KERs for entire data (a), KERG statistics for data sets of different categories (a-c).

KERG relationships retrieved for a particular keyword pair. Hence, the number of inputs equals the number of all possible combinations of two keywords (for a query with n keyword, this is $n!/2(n-2)!$). Since both the score aggregation functions in (6) and (7) are monotonic w.r.t. their element scores, the input entries can be indexed with and sorted by $\mathcal{S}(\langle k_i, k_j\rangle, e'_{\mathcal{K}}) * (\mathcal{S}(k_i, n'_{\mathcal{K}_i}) + \mathcal{S}(k_j, n'_{\mathcal{K}_j}))$. Instead of using the standard join operator in Algorithm 1, we can now apply rank join on these sorted inputs to compute top-k results.

8 Evaluation

We implemented the proposed approach in Java using JDK 1.6 on top of MySQL 5.1. The experiments were conducted on a commodity PC with 2.5-GHz Intel Core, 4 GB of RAM, and 500-GB HDD S-ATA II 7,200 rpm, running on Windows 7.

8.1 Data

The data used for the experiments are drawn from data sets prepared for the Billion Triple Challenge[1] (BTC). BTC data were crawled from major Semantic Web's websites during February/March 2009. BTC data were split into chunks of 10M statements each. All the chunks, additional information, and statistics are made available at http://vmlion25.deri.ie/index.html.

The data we used for the experiment are the chunk that can be found at http://vmlion25.deri.ie/btc-2009-small.nq.gz. The raw ungzipped file is 2.2 GB. This chunk of data contains a large number of data sources. Some of them contain less than 3K RDF triples. Excluding these small sources from the experiments resulted in a final data set that has about 10M RDF triples contained in 154 different sources.

Based on the number of RDF triples they contain, these sources can be classified into six categories. Table 2 shows statistics for each category and some example sources.

1. http://challenge.semanticweb.org/.

8.2 Data Preprocessing

Index Size and Building Time. During the index building process, we counted the number of keyword relationships, i.e., all pairs of keywords that are connected over a maximum distance d_{max}. This is to resemble the M-KS model [6], which captures all binary relationships between keywords. As discussed, E-KERG extends G-KS [7] to the keyword routing scenario. We counted the number of element-level keyword-element relationships (E-KERs) to capture this baseline. Finally, we consider the number of relationships in KERG (KERs). These numbers were counted for the entire data and separately for every subcategory. At various d_{max}, Fig. 6a illustrates the number of KRs versus E-KERs versus KERs for the entire data. Figs. 6b, 6c, and 6d show the number of KERs, the storage size required for the corresponding KERG indexes, and the time for building these indexes for data sets of different categories. Concrete numbers are provided in Table 3 for the entire data.

Expectedly, the number of KRs, E-KERs, and KERs increased with d_{max}. The increases were sharp when $d_{max} > 1$. Compared to E-KERs, the increases for KRs and KERs were moderate. As illustrated in Fig. 6a, the increase of KERs was comparable to KRs. Table 3 more precisely shows that for $d_{max} = 4$, the number of KERs was only about 20 percent higher than KRs while the number of E-KERs was more than three times higher. Thus, these results suggest that summarizing at the level of sets can substantially reduce the number of relationships.

TABLE 3
KERG Statistics for Entire Data

d_{max}	#KRs	#E-KERs	#KERs	Size (MB)	Time (min)
0	1,569,372	2,489,277	1,785,840	45.2	0
1	4,311,939	7,743,240	4,895,221	124.9	27.3
2	136,864,693	364,512,617	143,379,766	3,438.6	338.8
3	188,453,908	616,952,668	214,176,529	5,253.9	583.7
4	255,418,996	889,528,343	311,359,263	7,736.4	846.2

TABLE 4
Queries Employed in the Experiments

#	Keywords
Q1	Capital Washington
Q2	Barack Obama University
Q3	Film Titanic
Q4	ISWC Ontology Mapping 2004
Q5	ESWC2006 Semantic Search Publication
Q6	Ivan Herman W3C
Q7	Pascal Sebastian OWL Reasoning
Q8	Rudi AIFB ISWC2008
Q9	Studer AIFB Semantic Web
Q10	Semantic Web Publication
Q11	Andreas Harth Semantic Search Engine
Q12	Markus Denny Semantic Wikis
Q13	Information Retrieval Database 2006
Q14	Knowledge Management Conference
Q15	Beijing Conference Database 2007
Q16	Town River America
Q17	Paris Hotel
Q18	Karlsruhe Palace
Q19	Software Project 2003
Q20	Project AIFB 2005

Fig. 6b shows that also for every category of data sets, the number of KERs increased with d_{max}. This increase is reflected in the size of the index (Fig. 6c). For the entire data as well as for every category, the index size also increased with d_{max}. Likewise, the higher d_{max}, the higher was the amount of time needed for building the index (Fig. 6d). These results are not surprising because at a greater d_{max}, more relationships have to be taken into account.

However, we noted that these results were not strictly dependent on the data size. That is, the number of KERs, the size of the summary as well as building times did not directly correlate with the number of triples contained in the data sets. There were cases where relatively small data sets resulted in large KERGs. In Fig. 6d, for instance, we can see that times for categories of larger size were higher than those of smaller size. However, while category 2 was more than 150 percent larger in size, the difference in index building time to category 3 was less than 5 percent. In Figs. 6b and 6c, we can see that at $d_{max} = 4$, the number of KERs and the index size of category 2 were even smaller than those of category 3.

We found out that the dominant factor, which substantially determined index size and, thus, index building times, was the structural density of the data. In the experiments, densely structured data graphs resulted in higher building times than sparse graphs. Here, density refers to the distribution of edges within data sources and links between data sources. Category 3 for instance is relatively dense, containing data sets of smaller size that, however, exhibit large amounts of links to other sources, and contain several nodes that are well connected, i.e., reach hundreds of other nodes within $d_{max} = 4$.

8.3 Queries

Our main goals of the evaluation are to verify the validity and measure the relevance of the generated keyword routing plans. For a plan to be valid, the underlying query should produce answers. Further, interesting queries in this setting are those which combine results from different sources. We asked researchers who were familiar with the BTC data set to provide keyword queries that return meaningful results, along with descriptions of the intended information needs. In total, we have 30 keyword queries, each of them involves more than two data sources. One example submitted by participants is "Rudi AIFB ISWC2008," and the associated description is "Find the relationships between professor Rudi Studer, the AIFB Institute and the ISWC'2008 conference." The data sources containing partial answers to this query are uni-karlsruhe.de and semanticweb.org. The keywords of the first 20 queries are shown in Table 4.

8.4 Quality of Routing Plans

This part of the experiment investigates the quality of the routing plans. The goal is to find out whether they produce some results and how well they match the information needs, i.e., we want to assess both the validity and the relevance of routing plans. Recall that while KERG is built to capture all valid plans, the scoring mechanism is designed to focus on relevant plans. We now investigate how appropriately KERG summarizes the search space and how correctly the computed scores capture relevance.

Validity. Given a keyword query, let $[\mathcal{RP}]_k$ be the set of top-k routing plans returned by our approach and let $[\mathcal{RP}]^*$ denote the set of valid keyword routing plans. To find out whether a plan is valid, we looked into the data to see if it produces at least one result. Considering only the k best plans, we used the standard IR measure *precision at* k to compute the percentage of query plans that are valid out of k plans, i.e.,

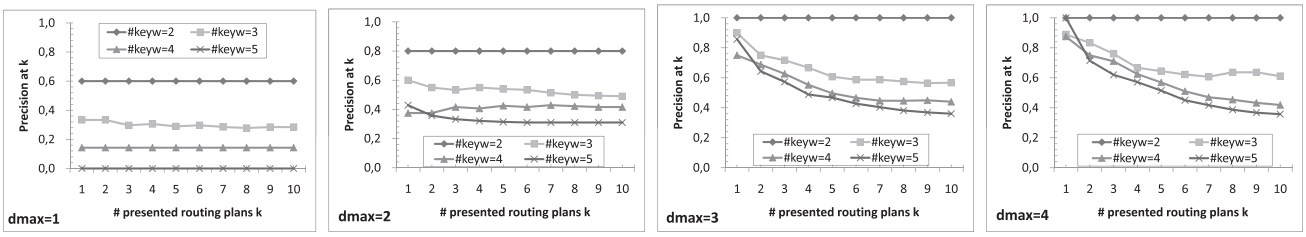
$$P@k = \frac{|[\mathcal{RP}]^* \cap [\mathcal{RP}]_k|}{|[\mathcal{RP}]_k|}.$$

Fig. 9a shows P@k at $k = 1$ up to 10, using the configurations $d_{max} = 0, 1, 2, 3, 4$. These values represent the average computed for all 30 queries. When only the single best plan was considered (P@1), precision up to 95 percent was achieved. Clearly, d_{max} had a positive effect on the quality of the produced plans. More valid plans were computed a higher d_{max}. When $d_{max} > 1$, precision was always higher than 50 percent at all values for k. However, we noted that precision did not increase substantially when $d_{max} > 3$, i.e., $d_{max} = 4$ improved $d_{max} = 3$ by only 7 percent at $k = 1$ and showed the same performance at $k > 6$.

Fig. 7 shows P@k obtained for different categories of queries at $d_{max} = 1, 2, 3, 4$. The goal of this illustration is to breakdown the results into queries of different complexities.

Clearly, P@k increased with d_{max}. Moreover, we can see that P@k correlated with query complexity. P@k values obtained for queries with a larger number of keywords were with only few exceptions, lower than queries with a smaller number of keywords. For instance, queries with only two keywords were processed with precision of 100 percent when $d_{max} > 2$. By contrast, queries with five keywords achieved high precision only when $d_{max} = 4$.

We noted that when $d_{max} = 1, 2$, P@k values were clearly different for queries with different number of keywords. However, at greater d_{max}, differences between query types were small. Also, P@k increased sharply at greater d_{max} (and smaller k). For instance, at $d_{max} = 4$, P@10 = 0.36, P@3 = 0.62, and P@1 = 1 for queries containing five keywords, and the difference in precision compared to queries containing four keywords is only 5 percent on average.



(a) $P@k$ for $d_{max} = 1$ (b) $P@k$ for $d_{max} = 2$ (c) $P@k$ for $d_{max} = 3$ (d) $P@k$ for $d_{max} = 4$

Fig. 7. $P@k$ for different categories of queries, at $d_{max} = 1, 2, 3, 4$ and $k = 1, 2, \ldots, 10$ (a-d).

To sum up, d_{max} had a positive effect, both the number of keywords and the number of presented results k had a negative effect; using a larger value for d_{max} minimizes the effect of query complexity and substantially improves precision, especially at a small k; $d_{max} = 3$ seems to be sufficiently large as no much improvement was achieved with $d_{max} > 3$. With $d_{max} = 3$, P@1 = 0.92 on average for all queries.

Relevance. To determine whether a valid plan also produces relevant results, we look at the descriptions of information needs provided by participants. Note that both the generated routing plans and the underlying routing graphs capture information needs, albeit at different levels of granularity. To work with a more precise representation, we used the routing graphs. To assess the relevance of a routing plan, we selected its routing graph that has the highest score and compared it with the description of the information need. When there is match, we assume that correct results matching that need can be obtained using a keyword search system and the routing plan is thus relevant.

A more direct way to measure relevance is to assess whether relevant plans actually "contain" relevant query results. However, this result-level relevance not only depends on routing but also on the (ranking performed by the) underlying keyword search system. We verify this using the keyword search system and ranking scheme presented in [16]. The relevant plans determined as discussed (mostly, there is only one) are used as inputs to this system, which employ them for pruning sources. With pruning, this system generates less results. However, we manually verified that more than 99 percent of these "missed" results are not relevant. Further, we observe that when using all the top-2 or top-3 plans for pruning, 6 percent or none of the results in the top-50 list are missed, respectively. Thus, we conclude that the plan(s) manually assessed as relevant may "contain" nonrelevant results but preserve all relevant results, and only the top-3 computed plans are needed to produce all the top-50 results.

As a measure for the relevance, we use the standard IR metric *reciprocal rank* (RR) defined as $RR = \frac{1}{r_{RP^*}}$, where r_{RP^*} is the rank of the plan $RP^* \in [RP]_k$ determined as relevant. If the relevant plan is not in $[RP]_k$, RR is simply 0. The Mean RR (MRR) is defined as

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{Q} RR.$$

Thus, MRR measures whether a relevant plan is in the top-k list and also, whether it is ranked high.

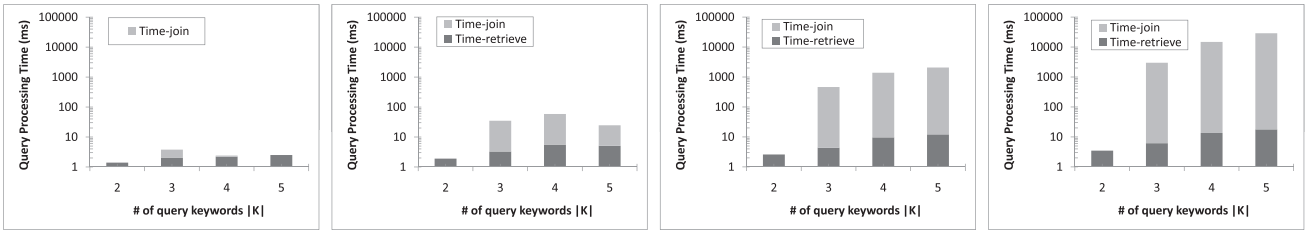
MRR values for queries of different categories at various d_{max} are shown in Fig. 9b. Clearly, also MRR were higher for queries with smaller number of keywords. When the number of keyword is 2 for instance, MRR was 0.6 for $d_{max} = 1$, up to 1 for $d_{max} > 2$. This means that for these queries, relevant plans were found and were also rank first. Also for queries with three or more keywords, results were reasonable. Average MRR for all queries was 0.86 for $d_{max} = 3$ and 0.96 for $d_{max} = 4$. All these values were obtained by looking for the relevant plan only in the list of top-5 results. Using $k = 1, 2, \ldots, 10$ did not yield substantial differences. For instance, MRR computed for the top-5 plans is only 1 percent worse than for the top-10 plans.

It seems that using $d_{max} = 3$ also yielded highly relevant plans: MRR $= 0.86$ means in this case that for all queries, a relevant corresponding plan could always be found and was ranked among the top 2.

Baseline. As pointed out in Section 7, the idea proposed for G-KS can be implemented by computing scores of KERG nodes and relationships based on all elements in the database (instead of considering only elements that participate in the set-level nodes and relationships captured by KERG). Further, scores of all nodes and relationships that have been retrieved for all query keyword pairs were taken into account (instead of considering only those that actually participate in the top-k routing graphs). Analogously, we adopt this ranking to compute the scores for E-KERG results (routing plans for E-KERG are computed using the same procedure as proposed for KERG). For $d_{max} = 3$, which is the best configuration in our experiment, P@1 for KERG and E-KERG, respectively, was only 0.38 and 0.35 (compared to 0.92) and MRR was only 0.32 and 0.43 (compared to 0.86) on average. These results suggest that considering scores not only at the keyword level but also their actual contributions to set-level elements and routing graphs that participate in the final answers can largely improve quality.

8.5 Performance of Plan Computation

Fig. 10a shows the average response times for computing routing plans. It depicts the performance for queries with different number of keywords $|\mathcal{K}|$ and also illustrates the effect of d_{max}. Expectedly, more time was needed as the number of query keywords increased. The increase was steady for $d_{max} = 3, 4$, and less so for $d_{max} = 1, 2$. The previous experiment showed that a higher value for d_{max} resulted in higher quality plans, but also larger indexes. The results of this experiment suggest that a larger number of keywords required more time to process—especially on large indexes. There is, thus, a tradeoff between result



(a) Time for $d_{max} = 1$ (b) Time for $d_{max} = 2$ (c) Time for $d_{max} = 3$ (d) Time for $d_{max} = 4$

Fig. 8. Times for retrieval and join for different query types, at $d_{max} = 1, 2, 3, 4$ (a-d).

quality and performance. Higher quality results were achieved at the cost of a larger index, which resulted in performance.

Overall, queries with two keywords were processed in less than 5 ms. All queries were processed within 90 ms when $d_{max} = 2$. In the previous experiment, $d_{max} = 3$ resulted in high-quality plans. This configuration showed also relatively good performance, required an affordable amount of time of 1 s on average. We now discuss results shown in Fig. 8 to provide a more detailed analysis at the level of query types and a breakdown into costs for retrieval and join.

Performance of Different Query Types. All 2-keyword queries (Q1, Q3, Q17, and Q18) were processed within 5 ms. This type of queries were processed efficiently even at large d_{max}. However, queries with more keywords could not be handled efficiently at $d_{max} \geq 3$. At $d_{max} = 4$ for instance, queries with more than two keywords needed several seconds up to one minute. Thus, while this setting produced results of highest quality, it is not really affordable in a typical web scenario demanding high responsiveness. More applicable is $d_{max} = 3$, which produced results in one second, while not compromising too much on quality.

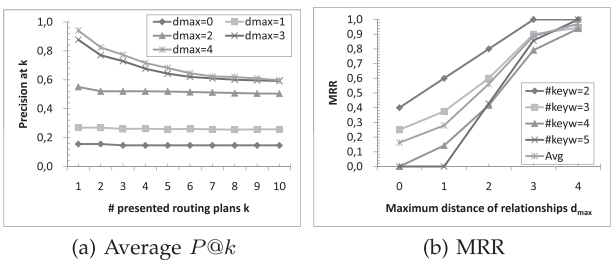
Besides the number of keywords, we found out that "popularity" was another factor. It seems that more time was need when query keywords have a large number of mentions, resulting in a large number of relationships that have to be processed. This was the case for Q10. All the three keywords $Semantic$, Web, and $Publication$ in Q10 are very popular, and thus, every keyword pair of this query produced a large number of matching relationships. In contrast, the keywords $Studer$ and $AIFB$ in Q9 have relatively few mentions. The time needed for Q9 is about one order of magnitude less than the time for Q10.

Breakdown into Retrieval and Join. Processing relationships means to retrieve data from the index and to join them. A breakdown of the time into the parts attributable to these two tasks are provided in Fig. 8. Costs for both retrieval and join increased with the number of keywords and d_{max}. Clearly, the reason for fast response time on 2-keyword queries is that they did not require join processing. Retrieval cost seemed to be relatively low. In all settings and for all queries, data were retrieved within 20 ms. The time needed for join largely increased with d_{max}. For $d_{max} > 2$, join processing made up more than 90 percent of the cost.

Baseline. We compared results obtained for KERG with results obtained for E-KERG. We have already discussed in Section 3 that E-KERG is a baseline that extends the G-KS [7] approach to the query routing problem. Results obtained for different configurations of d_{max} suggest that KERG outperformed E-KERG by several order of magnitudes. For $d_{max} = 3$ for instance (shown in Fig. 10b), 50 percent of the queries with two keywords or more needed more than 6 mins (= timeout) when using E-KERG, while with KERG, maximum time was only several seconds and average time was 1 s.

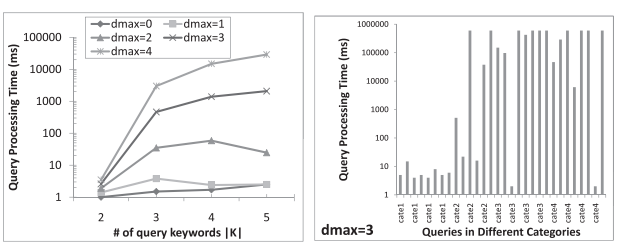
8.6 Keyword Search with/without Routing

Keyword query routing can be employed when the subject of interest is not necessarily results but sources that match some information needs. After selection, these sources can be preprocessed (cleaned) for improving the quality/efficiency of keyword search as well as used for other purposes. In the context of keyword search, it enables a new paradigm where instead of considering all, only relevant combinations of sources are considered. We study this effect of routing on keyword search in this experiment. We used a fast keyword search system that is optimized for time performance using materialized indexes of paths and subgraphs [16]. It finds answers for keyword queries using all sources (*KS*). This system does not compute all but uses several (ranking) mechanisms to prune some answers. We verify that the maximum number of answers was 50, i.e., it produces top-50 results.



(a) Average $P@k$ (b) MRR

Fig. 9. Validity of the plans measured using $P@k$ and relevance measured using MRR.



(a) Time KERG (b) Time E-KERG

Fig. 10. (a) Average times for KERG, (b) times for E-KERG at $d_{max} = 3$.

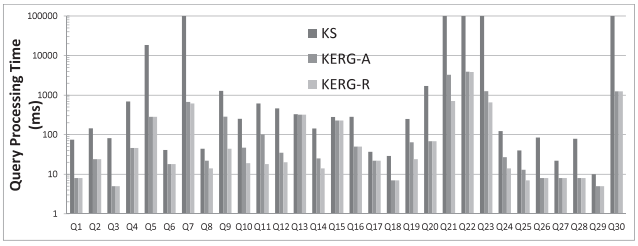


Fig. 11. Performance for KS with/without routing.

We analyze the cost of this by comparing KS's performance with the same system, which however, runs under two other configurations. 1) In the first configuration, KS prunes all sources but the ones captured by the routing plan computed using KERG that has been determined as relevant (*KERG-R*); thus, while KS delivers all results, KERG-R returns only the ones matching the information needed. Then, we verified that a maximum number of top-3 plans is needed to compute all top-50 results; accordingly, in the second configuration, KS uses sources captured by the top-3 plans (*KERG-A*). The results produced by KERG-A and KS are the same.

Fig. 11 shows the performance results of these systems over the 30 queries. The times for KERG-A and KERG-R include both the time for routing and for computing keyword search results. We noted that w.r.t. queries with five keywords, KS could process only 1, while the other five queries hit time-out limit of 100 s. Not counting these five queries, KERG-A is 15 times faster while KERG-R is 20 times faster than KS on average. Thus, routing has a large positive effect on the performance of keyword search. Further, the results suggest that keyword search without routing is especially problematic when the number of keywords is large. Hence, routing can be seen as a promising alternative paradigm especially for cases, where the information need is well described and available as a large amount of texts.

9 Conclusion

We have presented a solution to the novel problem of keyword query routing. Based on modeling the search space as a multilevel inter-relationship graph, we proposed a summary model that groups keyword and element relationships at the level of sets, and developed a multilevel ranking scheme to incorporate relevance at different dimensions. The experiments showed that the summary model compactly preserves relevant information. In combination with the proposed ranking, valid plans (precision@$1 = 0.92$) that are highly relevant (mean reciprocal rank $= 0.86$) could be computed in 1 s on average. Further, we show that when routing is applied to an existing keyword search system to prune sources, substantial performance gain can be achieved.

References

[1] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-Style Keyword Search over Relational Databases," *Proc. 29th Int'l Conf. Very Large Data Bases (VLDB),* pp. 850-861, 2003.

[2] F. Liu, C.T. Yu, W. Meng, and A. Chowdhury, "Effective Keyword Search in Relational Databases," *Proc. ACM SIGMOD Conf.,* pp. 563-574, 2006.

[3] Y. Luo, X. Lin, W. Wang, and X. Zhou, "Spark: Top-K Keyword Query in Relational Databases," *Proc. ACM SIGMOD Conf.,* pp. 115-126, 2007.

[4] M. Sayyadian, H. LeKhac, A. Doan, and L. Gravano, "Efficient Keyword Search Across Heterogeneous Relational Databases," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE),* pp. 346-355, 2007.

[5] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding Top-K Min-Cost Connected Trees in Databases," *Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE),* pp. 836-845, 2007.

[6] B. Yu, G. Li, K.R. Sollins, and A.K.H. Tung, "Effective Keyword-Based Selection of Relational Databases," *Proc. ACM SIGMOD Conf.,* pp. 139-150, 2007.

[7] Q.H. Vu, B.C. Ooi, D. Papadias, and A.K.H. Tung, "A Graph Method for Keyword-Based Selection of the Top-K Databases," *Proc. ACM SIGMOD Conf.,* pp. 915-926, 2008.

[8] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword Search in Relational Databases," *Proc. 28th Int'l Conf. Very Large Data Bases (VLDB),* pp. 670-681, 2002.

[9] L. Qin, J.X. Yu, and L. Chang, "Keyword Search in Databases: The Power of RDBMS," *Proc. ACM SIGMOD Conf.,* pp. 681-694, 2009.

[10] G. Li, S. Ji, C. Li, and J. Feng, "Efficient Type-Ahead Search on Relational Data: A Tastier Approach," *Proc. ACM SIGMOD Conf.,* pp. 695-706, 2009.

[11] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional Expansion for Keyword Search on Graph Databases," *Proc. 31st Int'l Conf. Very Large Data Bases (VLDB),* pp. 505-516, 2005.

[12] H. He, H. Wang, J. Yang, and P.S. Yu, "Blinks: Ranked Keyword Searches on Graphs," *Proc. ACM SIGMOD Conf.,* pp. 305-316, 2007.

[13] 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," *Proc. ACM SIGMOD Conf.,* pp. 903-914, 2008.

[14] T. Tran, H. Wang, and P. Haase, "Hermes: Data Web Search on a Pay-as-You-Go Integration Infrastructure," *J. Web Semantics,* vol. 7, no. 3, pp. 189-203, 2009.

[15] R. Goldman and J. Widom, "DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases," *Proc. 23rd Int'l Conf. Very Large Data Bases (VLDB),* pp. 436-445, 1997.

[16] G. Ladwig and T. Tran, "Index Structures and Top-K Join Algorithms for Native Keyword Search Databases," *Proc. 20th ACM Int'l Conf. Information and Knowledge Management (CIKM),* pp. 1505-1514, 2011.

Thanh Tran was a consultant and a software engineer at IBM and Capgemini, and served as a visiting assistant professor at Stanford University. He has been mainly working on the topics of semantic data management, keyword search on semantic data, and semantic search. His interdisciplinary work is published in more than 40 top-level journals and conference proceedings and earned the second prize at the Billion Triple Challenge 2008 and the best Linked Data paper at ESWC 2011.

Lei Zhang received the bachelor's degree in computer science from Tongji University, China. He graduated from Karlsruhe Institute of Technology (KIT), Germany, with a diploma's degree in Informatics. He is currently working toward the PhD degree at the AIFB institute of KIT, where he is working on the project XLike. He has been involved in many research projects on semantic search.