STM JOURNALS

Journal of
Web Engineering & Technology

ISSN: 2455-1880
Volume 11, Issue 2, 2024
May–August
DOI (Journal): 10.37591/JoWET

https://journals.stmjournals.com/jowet

Research JoWET

Optimizing Keyword Querying on Structured Knowledge Bases

Ashokkumar Gurusamy[1,*], Vinod Kumar[2], Piyush Sharma[3]

Abstract

As the digital landscape burgeons with unstructured information, the need for structured knowledge bases becomes paramount. The Semantic Web employs Resource Description Framework and SPARQL as a cornerstone in organizing information. This work explores keyword querying approaches on knowledge bases such as Freebase, emphasizing their usability challenges, particularly in complex queries. Existing approaches, while valuable, exhibit limitations in handling intricate queries with multiple joins. The proposed Entity Relationship Keyword Querying framework addresses this shortfall, enhancing usability and capability. This research outlines the Resource Description Framework/SPARQL foundations, surveys related works, presents Entity Relationship Keyword Querying, and discusses its potential enhancements. The work concludes with a prototype implementation summary, paving the way for future refinements in keyword querying on extensive knowledge bases.

Keywords: Semantic web, entity relationship keyword querying, ERKQ, resource description framework, RDF, SPARQL, prototype implementation, keyword querying

INTRODUCTION

While the web is undoubtedly the largest collection of human knowledge, it is largely constituted of unstructured information—web pages with text, images, and other media embedded within them. This organization (or the lack of) of information poses certain difficulties and inherent limitations in building automated systems that can consume and process information.

To remedy some perceived challenges, there is an ongoing effort to build and maintain structured knowledge bases. As of today, there exist several such knowledge repositories, for example, Freebase, DBpedia, Wikidata, and so on. Apart from those that are freely accessible, several internet businesses like Google exist. Yahoo! and Baidu maintain their proprietary knowledge bases, successfully assisting internet web search applications.

***Author for Correspondence**
Ashokkumar Gurusamy
E-mail: agashokkumar@gmail.com

[1]Principal Software Engineer, Fidelity Investments, United States
[2]Vice President, GAVS Technologies, New Jersey, United States
[3]Associate Developer, Groupsoft Us Inc, United States

Received Date: June 26, 2024
Accepted Date: July 23, 2024
Published Date: August 02, 2024

Citation: Ashokkumar Gurusamy, Vinod Kumar, Piyush Sharma. Optimizing Keyword Querying on Structured Knowledge Bases. Journal of Web Engineering & Technology. 2024; 11(2): 28–50p.

In the wake of the increasing popularity of structured data and to provide a common platform for data exchange, the World Wide Web Consortium decided to collect these efforts under the umbrella of the Semantic Web. The Semantic Web is an extension of the Web through standards that promote common data formats and exchange protocols on the Web, most fundamentally the Resource Description Framework (RDF).

Even though the attempt is toward giving structure to data, the traditional concepts of relational schema and relational data model are eschewed. The data is most commonly modeled as a graph of entities and objects, with an

accompanying schema and an ontology. This means that well-developed ideas of querying on relational databases can no longer be directly applied, and toward this end, query languages such as SPARQL and corresponding query processing engines have emerged.

An important feature of modern knowledge bases is their vastness. Knowledge bases with as many as a billion facts are widely available. While SPARQL and other structured query languages have some strong points, their usability is hindered by the limited schema knowledge available to an end user. Therefore, approaches akin to keyword querying in databases that abstract out schema requirements from the user have become increasingly relevant.

This work will survey some such keyword querying approaches on structured knowledge bases. We shall also highlight a common weakness: insufficient performance on complex queries involving several joins. Our approach toward keyword querying will address this limitation of existing approaches.

OUTLINE

Before delving into our discussion, we offer an outline of RDF/SPARQL specifications and associated notions. Following this, we review a handful of relevant works that share a similar rationale to ours. We highlight important ideas that can be incorporated into building future systems. They shall provide a detailed discussion of our proposed approach to the problem. We conclude with brief notes on our prototype implementation, elaborating on the various choices we had to make. Adds a final page of summary and directions for future work.

BACKGROUND

In this section, we cover the necessary background required for appreciation of this work and the area in general. We begin by describing two defining elements of the Semantic Web: (1) Resource Description Format (RDF) that allows a flexible representation of structured information, and (2) the SPARQL interrogation syntax employed for fetching and handling information stored in RDF. Toward the closing, we introduce the Freebase knowledge base—the structured data source that shall be considered primary for the remaining part of this work.

RESOURCE DESCRIPTION FORMAT

Before we dive headfirst into the territory of the semantic web, we must first understand how it stores data. We start from the ground up by outlining the basic idea of conceptualizing data as a graph—an approach that is in many ways contrasted to the relational model assumed by traditional databases.

The notion of an entity is the fundamental building block of knowledge graph (as we shall call it when emphasizing its graph structure). In most applications modeling general purpose real-world information, an entity has a separate and distinct existence and objective or conceptual reality. To make it more concrete, here are a few examples of entities:

- Indian Institute of Technology, Bombay—an educational institute.
- Mumbai—a city
- An Evening in Paris—a Bollywood film
- Sania Mirza—a person

To put it simply, a structured knowledge base captures information about entities. Entities can be related to each other and can have properties (or attributes). The idea of expressing relations between entities is directly borrowed from the notion of the Entity-Relationship Model.

The RDF serves as a structure to depict details regarding entities in a graphical format. It formalizes the notion of entities and relationships between them and the graph thus produced.

An RDF-triple is a 3-tuple of (subject, predicate, object). While the subject is typically an entity, the object can be an entity or a non-entity. An RDF object is allowed to be an arbitrary string, which makes it flexible in capturing information such as dates, addresses, and other metadata information that a subject (entity) can be associated with. An RDF predicate embodies the notion of a connection that delineates the essence of the linkage between the respective subject and object.

Since the RDF evolved around WWW, it encodes the subject, predicate, and, at times, the object as Uniform Resource Identifiers (URI). An RDF document consists of RDF triples, typically one triple per line. Figure 1 illustrates an example RDF graph that the following RDF document represents:

```
1: <http://www.example.org/~joe/
contact.rdf#joesmith>
      <http://www.w3.org/1999/02/
      22-rdf-syntax-ns#type>
      <http://xmlns.com/foaf/0.1/Person>
2: <http://www.example.org/~joe/
contact.rdf#joesmith>
      <http://xmlns.com/foaf/0.1/homepage>
      http://www.example.org/~joe
3: <http://www.example.org/~joe/
contact.rdf#joesmith>
      <http://xmlns.com/foaf/0.1/mbox>
      mailto:joe.smith@example.org
4: <http://www.example.org/~joe/
contact.rdf#joesmith>
      <http://xmlns.com/foaf/0.1/givenname>
      Joe
5: <http://www.example.org/~joe/
contact.rdf#joesmith>
      <http://xmlns.com/foaf/0.1/family_name>
      Smith
```

RDF predicates are labeled arrows that connect an RDF subject (at their tail) to an RDF object (at their head).

Many of the contemporary freely accessible knowledge bases support their distribution in RDF format. While an RDF document is a natural way of representing and storing graphs, it is seldom useful in this form. Several RDF databases have emerged that allow the loading of RDF documents and support APIs that can be used to query and use the data. Traditional query languages like SQL are not well suited for querying RDF data; thus, query languages and query processing engines are needed to target this format. SPARQL is one such query language that has been successful, and we shall now present a brief overview of it.

SPARQL PROTOCOL AND RDF QUERY LANGUAGE

SPARQL [1], standing for the SPARQL Protocol and RDF Query Language, functions as a semantic inquiry tool specifically designed for RDF databases. It empowers users to craft queries targeting data that can be characterized as “key-value” or, more precisely, adheres to the RDF specification set by the W3C.

Though we earlier presented the RDF data model as a graph, it can also be thought of in terms of the SQL relational model as a table with three columns, one each for the subject, the predicate, and the object. As we noticed earlier, the data in the object column can be generally heterogeneous, which contrasts with relational databases. The predicate value usually implies the column data type or may be specified separately in an ontology.

As another alternative, all triples belonging to a particular subject may be represented as a single table row where the subject is the primary key. A column for each possible predicate may be stored, with the value in a cell storing an RDF object for (row, column) pair of (subject, predicate). However, as is seen in real-world datasets, a column could contain multiple values (like the column "family members") for the same row key.

SPARQL offers a comprehensive array of analytical query functionalities, including JOIN, SORT, and AGGREGATE operations, tailored for datasets where the schema is inherently embedded within the data, eliminating the need for a separate schema definition. Typically, schema information, or ontology, is provided externally to facilitate unambiguous joins between different datasets. Moreover, SPARQL presents specialized syntax for traversing graphs within the data.

Example Query

We shall now illustrate the basic syntax and features of SPARQL using a simple query on the data of Figure 1.

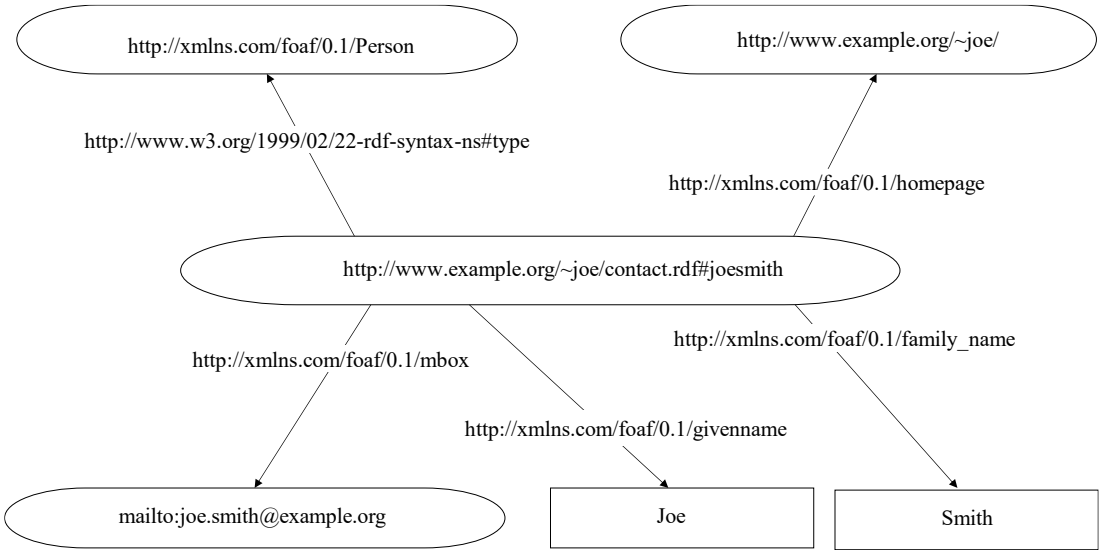

Figure 1. An example RDF graph.

The SPARQL query for retrieving (person, homepage) pairs for all persons in the data is:

```
PREFIX foaf:   <http://xmlns.com/
foaf/0.1/>
PREFIX w3: <http://www.w3.org/1999/
02/22-rdf-syntax-ns>
FROM foaf:graph

SELECT ?person ?homepage WHERE {
?person w3:type foaf:Person,
?person foaf:homepage ?homepage
}
ORDER BY ?person
```

Since RDF subjects and predicates are represented as URIs, they tend to have common prefixes. To prevent repetition of these prefixes in the query, the PERFIX keyword has been used in the above query. In general, an RDF database might contain multiple RDF graphs. The from keyword allows a user to specify a particular graph to be queried.

The SELECT clause asks to specify variables matched by the triple patterns inside the WHERE clause. A pattern comprising three elements, known as a triple pattern, represents an RDF triplet in which one or more components have been substituted with placeholders.

It is worth noting that triple patterns allow a natural way of expressing joins without an explicit schema, which makes SPARQL more powerful than storing RDF in a relational schema and running SQL queries on it, as was previously discussed.

SPARQL has been implemented in several programming languages and graph databases. In particular, the Apache Jena Framework [2] and Openlink Virtuoso [3] are widely used implementations of the standard.

FREEBASE KNOWLEDGE BASE

A major prerequisite for the Semantic Web's success is the availability of large amounts of structured data. The effort of building and maintaining comprehensive datasets is largely divided in purpose and content by the organizations supporting these projects. On the one end, there are freely accessible knowledge bases such as DBpedia, Wikidata, and Freebase, and on the other, there are closed-source projects such as Google Knowledge Graph and the Wolfram Knowledgebase.

We decided to stick to Freebase as our underlying data source for most of our work. The choice demands us to tweak our approaches slightly to use the Freebase schema. Still, the ideas presented are sufficiently general and can be adapted to any of the popular knowledge bases.

Freebase was a large collaborative knowledge base with data curated mainly by its community members. It first appeared in 2007 and was originally developed by the company Metaweb (en.wikipedia.org/wiki/Metaweb). Though the project is no longer active, the complete data is available freely through Google Developers (developers.google.com/freebase).

Freebase encompasses tens of millions of subjects, thousands of categories, and tens of thousands of attributes. The Freebase graph comprises over a billion pieces of information, commonly referred to as RDF triples. The magnitude of this can be appreciated compared to Wikipedia, which has less than 5 million English articles.

Schema

The nodes within the Freebase graph are denoted using /type/object/, while the edges are represented by /type/link/.

Freebase boasts over 39 million topics covering a myriad of real-world entities such as individuals, locations, and objects. These topics serve as the nodes within the graph, though not every node necessarily represents a topic.

Examples of the diverse range of topics found in Freebase include: label=

- Physical entities like Bob Dylan, the Louvre Museum, and the planet Saturn.
- Artistic and media creations such as The Dark Knight (film) and Hotel California (song).
- Classifications such as noble gas and Chordate.
- Abstract concepts like love.
- Schools of thought or artistic movements like Impressionism.

Each Freebase topic can be viewed from various perspectives. For instance, Bob Dylan is not just a musician but also a songwriter, singer, author, and film actor. To capture this multi-faceted nature, Freebase defines the types assigned to each topic. These types encompass different sets of properties pertinent to the topic. For example: label=

- The music artist type includes properties listing Bob Dylan's albums and the musical instruments he played.

- The book author type encompasses properties listing his written works and associated literary movements.
- The company type includes numerous properties detailing a company's structure, financials, and products.

Thus, types serve as conceptual containers housing properties essential for describing specific aspects of information. Further details regarding the Freebase schema will be provided as needed in subsequent sections.

SURVEY OF EXISTING WORK

This section aims to give an overview of the existing work that tackles the problem of querying structured knowledge bases. As elaborated in the initial section, the utilization of organized inquiry dialects faces constraints due to the necessity for the user to possess knowledge of the structure of the data under examination. We will see multiple approaches that seek to remedy that in this section. A full coverage and critique of all approaches is beyond the scope of this report, so we shall select a few interesting ones for the purpose of this section. The approaches will be organized by their central ideas.

FREE-KEYWORD QUERYING

In [4], the authors introduce a retrieval model designed for searching RDF graphs using keywords. Unlike methods that retrieve entity tuples, this model returns a ranked list of RDF subgraphs, enabling a holistic treatment of triples by considering the relationships between entities.

For handling keyword inquiries across RDF graphs, every triple t_i is linked with a collection of terms extracted from its subject and object and indicative terms for the predicate, shaping a document D_i. The terms in D_i are stemmed and stored in an inverted index alongside the term frequency $c(w, D_i)$.

The first phase in query processing commences with fetching subgraphs that correspond with the user' s keyword inquiry. Two restrictions are enforced to guarantee the retrieval of compact and pertinent subgraphs:

1. uniqueness and maximality of subgraphs, and
2. inclusion of triples matching different sets of keywords to avoid redundancy.

Given a keyword query $q = q_1, q_2, \ldots, q_m$, the subgraph retrieval algorithm utilizes the inverted index to retrieve lists $L_1, L_2, \ldots, L_m$. The set E of all unique triples in these lists forms a query graph, which is analyzed using an adapted network motif detection algorithm.

Subsequently, retrieved subgraphs are ranked based on a statistical language model. Every subset of connected nodes $G = \{t_1, t_2, \ldots, t_n\}$ is evaluated based on the likelihood of producing the inquiry q according to the linguistic structure inherent in G, assuming that the terms in the inquiry are statistically independent. This likelihood $Pr(q|G)$ is computed as the product of individual term probabilities $Pr(q_i|G)$, where $Pr(q_i|G)$ represents a composite framework composed of linguistic structures.

The performance of this model is compared with existing techniques for keyword search over structured data [5, 6]. For further details, we refer readers to the original paper [7].

CONVERTING QUERIES TO SPARQL

DEANNA [8] introduces a framework for answering natural language questions using structured knowledge bases. It involves converting the question into a SPARQL query for processing. The main challenge is disambiguating user intent within the question.

The framework first identifies semantic items in the question using specialized detectors for classes, entities, relations, and interrogative pronouns.

Then, it detects triploids, which are 3-token tuples representing relations and their arguments. These triploids, along with candidate phrases, are combined into q-units, which are phrase-level triples.

The next critical step is disambiguation, where a weighted disambiguation graph resolves the mappings of phrases to semantic items. This graph, illustrated in Figure 2 [9], is tripartite and incorporates weights based on both semantic coherence and syntactic similarity. Q-nodes connect phrase triples, as indicated by dotted edges. Disambiguation aims to identify a dense subgraph in the disambiguation graph, with edge weights reflecting both syntactic similarity and semantic coherence. Constraints ensure the subgraph represents valid triples. An integer linear program encodes the objective and constraints, which an ILP solver then solves to find the subgraph. Once disambiguation is complete, semantic grouping occurs, forming semantic triples. As an illustration, if the relationship "marriedTo" links the individual indicated by "Who" and the "writer," it establishes the semantic trio "person marriedTo writer." These trios undergo conversion into SPARQL inquiries using a method grounded in rules. For instance, "person marriedTo writer" evolves into "?x type person," "?x marriedTo?w," and "?w typewriter."

QUERYING BY EXAMPLE ENTITY TUPLES

GQBE [10] introduces a method for querying knowledge graphs using example entity tuples, with both user input and output being entity tuples referred to as query tuples and answer tuples, respectively. In essence, when presented with a data diagram G and a query tuple t, the objective of GQBE is to identify the top-k response tuples t' that possess the greatest resemblance, as measured by their similarity metrics $score_t(t')$.

The comparison measure $score_t(t')$ is established by assessing the connections between entities in t and t'. Rather than matching entities individually in t, a neighborhood graph is constructed from t, capturing relevant features. The score calculation entails corresponding two diagrams created from t and t'.

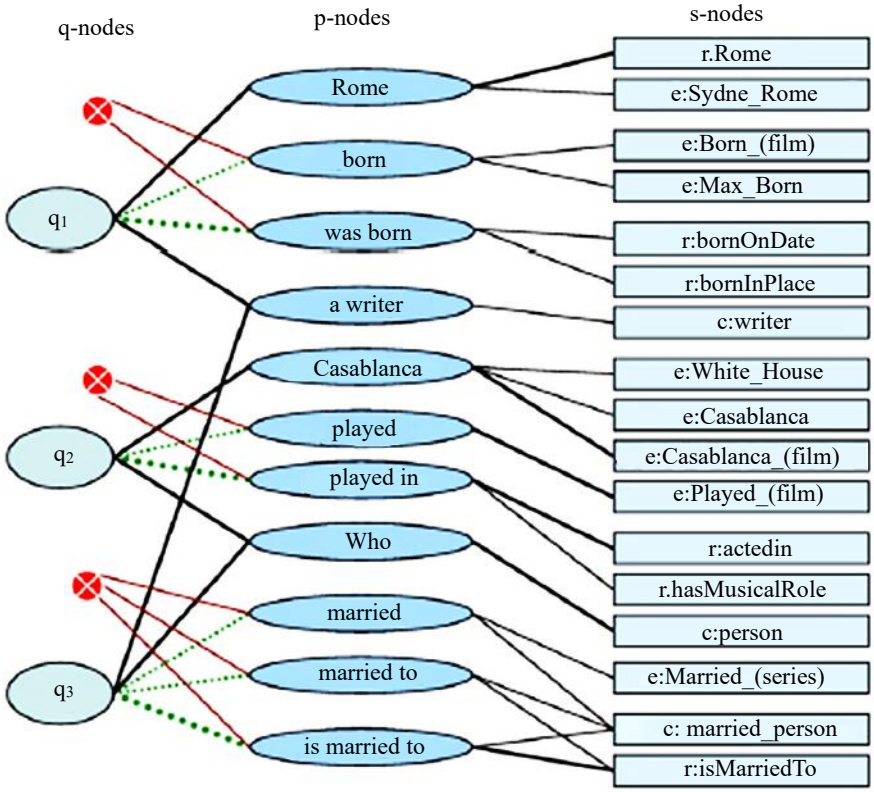

Figure 2. Disambiguation graph. Attribution: [9].

The neighborhood graph H_t of a query tuple t is a weakly connected undirected subgraph of G consisting of nodes reachable from t nodes by paths of length less than a user-defined threshold d. Nonetheless, H_t may grow disproportionately even with minor d values, creating a Maximum Query Graph (MQG) derived from H_t. MQG_t, with a parameter m, is a subgraph of H_t maximizing total edge weight while containing all t nodes and at most m edges.

Finding MQG_t for a given m is proven to be NP-hard, leading GQBE to employ a greedy algorithm for an approximate solution.

With MQG_t established, the objective shifts to finding answer graphs similar to MQG_t. An answer graph A to a query graph Q is a subgraph of G edge-isomorphic to Q. Similarity is measured via a composite of content and structure scores. The arrangement metric measures the notable arrangement within MQG_t as depicted by Q, whereas the substance rating acknowledges indistinguishable elements within corresponding elements in A and Q.

At last, GQBE scours the graph G for the top-k solution subgraphs using similarity evaluations in a prioritized fashion, though the specific algorithm is omitted from this discussion.

INTERACTIVE QUERIES

FreeQ [11] is an interactive interface for refining queries incrementally. It allows users to start with basic keywords and progressively refine them into structured queries. Figure 3 illustrates the interface. When users input a keyword query like "Tom Hanks Terminal," FreeQ presents the top-k structured query options. If the correct query isn't immediately clear, users can use interaction options to refine their intent. Each interaction narrows down the available queries until the user finds the desired one or requires further options.

Selecting interaction options yielding the highest information gain is crucial, revealing as much about the user's intent as possible. FreeQ achieves this by leveraging hierarchical conceptual models atop the database schema and external ontologies like WordNet and YAGO.

Generating the interpretation space is a prerequisite before obtaining the top-k interpretations. However, directly materializing this space for large knowledge bases like Freebase is impractical. Therefore, FreeQ adopts a query hierarchy introduced in previous work, incrementally growing it during the user interaction phase to generate the most probable query construction options aligning with the user's selections at each step.

ENTITY RELATIONSHIP KEYWORD QUERIES

This section presents a detailed discussion of our approach to supporting keyword queries. The earlier part of the section introduces the query model illustrated with examples. A brief overview of the query processing mechanism follows. The latter half of the section focuses on query interpretation—a theme central to our approach. Finally, the discussion is closed with a description of how the result set of the query is produced.

XIII. QUERY MODEL

The notion of Entity Relationship Query (ERQ) or Shallow Semantic Query (SSQ) is defined by [12], [13] as a powerful and flexible query form that allows users to specify expected entity types or categories along with selection predicates for entities and relation predicates for a relationship between entities.

For example, a query described as "Find PERSON near 'Stanford graduate', and COMPANY near 'Silicon Valley', where PERSON founded COMPANY" can be described in ERQ form as in Figure 4.

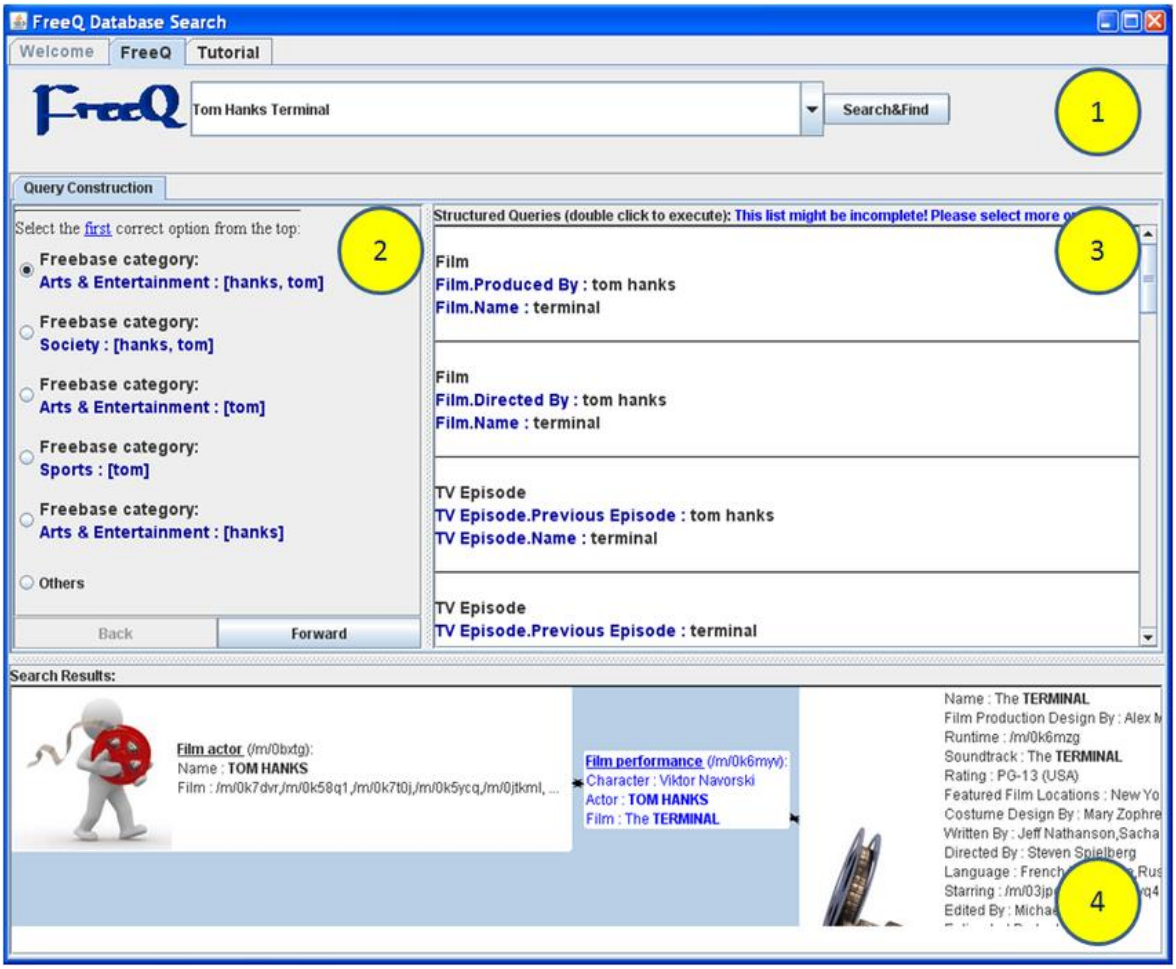

Figure 3. FreeQ GUI. The components of the FreeQ GUI includes (1) an input field for a keyword query, (2) interaction options, (3) top-k structured queries, and (4) query results. Attribution: [11]

```
SELECT x, y
FROM PERSON x, COMPANY y
WHERE x:['Stanford', 'graduate']
WHERE y:["Silicon Valley"]
AND x,y:['found']
```

Figure 4. An example ER Query.

The query of Figure 4 seeks the entity pairs (x, y) such that entity x is of type PERSON (and further a Stanford graduate), entity y is of type COMPANY (in particular, a Silicon Valley company) and x has founded y. x and y are entity variables that will bind to entities in the query result. PERSON and COMPANY describe the category/types of these entities. “Stanford graduate” and “Silicon Valley” play the role of selector keywords for entities x and y, respectively. Finally, found is the relation keyword describing the relationship between entities x and y.

To make our discussion convenient, we formalize the notion of ERQ as follows. Our system expects the user to specify:

- The number of entities (n) desired in a result. The entity variables will be referred to as e_1, e_2, . . . , e_n.
- A list of category keywords C_i for each entity variable x_i.
- A list of selector keywords K_i for each entity variable x_i.

- A list of relation keywords R_{ij} describing the relationship between entities x_i and x_j for all $I \neq j$.

The query result shall be one or more n-tuples, where n is as specified in the query. In particular, the elements of the tuple bind to entity variables (e_1, e_2, . . . , e_n).

The category keywords are used to specify what category or type the corresponding entity falls in. Selector keywords desire that the corresponding entity be associated with these words. Relation keywords specify the relation predicates in the same manner as ERQ. It is worth noting that the user may leave each of C_i, K_i, and R_{ij} empty (unspecified). The user is also not expected to specify these query keywords in conformity with the schema of the queried RDF graph.

We now illustrate another example to ease understanding of our ERQ formalism. Suppose we want to find the parent-child pairs of U.S. presidents who have attended the same university. Such a query demands three entities e_1, e_2, and e_3—e_1, e_2 are the presidents, and e_3 are their common alma mater. A user may formulate the query in the following manner:

- Number of entities is 3.
- C_1 ={'person'}, C_2 ={'person'}, and C_3 ={'university'}.
- K1 ={'US', 'President'}, K2 ={'US', 'President'}, and K3 ={}.
- R12 ={'parent'}, R13 ={'education'}, and R23 ={'education'},

As we observed in Section 3, the problem of segmenting a user keyword query into the keywords for the type/categories of entities sought, and those for describing the relationship between these entities inherently limits the performance (in both precision and recall) of the querying system. The ER query model might seem like a convenient ignorance of such issues, and in many ways, it is. Our stance is that ER queries do not expect the amount of user sophistication that SPARQL querying systems expect. It is a sweet compromise between SPARQL querying and free-keyword querying that allows us to focus on challenges beyond query segmentation. Further, we believe that our ideas in this work can be supplemented with a layer of query segmentation—enabling free-keyword querying.

Query Graph

Any given ER query can be thought of as edge-labeled directed graph G = (V,E), with the vertex set V ={e_1, e_2, ..., e_n} and the labeled edge set $E = \{(e_i \xrightarrow{R_{ij}} e_j) \mid \forall\ R_{ij} \in \text{query}\}$. Answering the query would then mean finding subgraphs in the data that match the query graph. In this direction, the ER keyword query model can be viewed as a relaxed variant of SPARQL query (The ER keyword query model is still very basic compared to SPARQL—which supports operations like sorting and aggregation that we do not consider here. Extensions of the ER keyword query model to support aggregation are left as future work), in which keywords can be used in place of actual schema elements of the data. While we do not explicitly use the graph interpretation of the query, except when a join is required to produce the final results, it is still an important aid in visualizing the overall query processing mechanism.

QUERY PROCESSING

The central goal of our work is to enable users to query structured knowledge bases without expecting any schema knowledge. In the context of the previous ER query model, this means that keywords supplied in a user query will not generally match those used for data representation in the underlying data source. For example, a user might use the relation keyword "established" to mean "establishment" of a company or an organization, but in the data source, the actual predicate sought could be "founded"—which roughly means the same as "established" in the case of companies.

Therefore, to achieve superior recall, we must explore many possible meanings of a keyword the user uses. Furthermore, a user-specified keyword could be ambiguous because it could mean any of several entities or predicates in the data. To achieve superior precision, we must be able to disambiguate the intent of the user query.

Our approach to the above two problems is to ask the user for help. Specifically, our inquiry processing mechanism will request the user to choose a portion of intermediate "potential" entities or predicates that the system could examine while query processing.

Overall, the query processing proceeds with the repeated execution of two steps:

1. Predicate selection and
2. Entity selection.

Predicate selection

As we assume that the user has no familiarity with the representation of the underlying data, the relation keywords R_{ij} specified by the user cannot generally identify a unique predicate in the RDF graph. Therefore, in this stage, the system presents a set of candidate predicates that match the given relation keywords R_{ij}. The user is then expected to select a subset of these predicates that align with her intent. It is to be ensured that a small yet accurate enough set of candidate predicates is presented to the user to minimize user effort. This step, predicate selection for p_{ij}, is performed for each of the relations $x_i \sim x_j$ specified initially by the user.

Entity selection

After the user has selected the desired set of predicates $\{p_{ij}^1, p_{ij}^2, \ldots, p_{ij}^r\}$ that comply with the given relation keywords R_{ij}, the system shall try to determine the entities x_i and x_j obeying any of these relations.

The RDF graph is queried to find all entities x_i and x_j that match the triple pattern $\langle ?x_i, p_{ij}^k, ?x_j \rangle$ (k = 1, . . . , r). By the sets X_i and X_j, we mean the set of candidate entities thus obtained for entity variables x_i and x_j, respectively. These sets X_i and X_j are further filtered, based on the category keywords C_i, C_j, and selector keywords K_i, K_j. The resulting sets $\widetilde{X}_i$ and $\widetilde{X}_j$ obtained after filtering are presented as candidate entities for entity variables x_i and x_j.

At this step, the user is expected to select a subset of these entities that align with her intent. As in the case of predicate selection, it is to be ensured that a small enough yet accurate set of candidate entities is presented to the user. The entity selection step for entity variables x_i and x_j is performed after the predicate selection step for p_{ij}.

Once the user agrees on a set of candidate entities $\tilde{X}_i$ for entity variable xi, this information will reduce the complexity f the further entity selection stages. This process terminates when predicate and entity selection have been performed for each predicate and entity in the query. The entity-relation graphs are finally ranked and returned to the user as query results.

QUERY INTERPRETATION

Even when we depend on the user's input for query disambiguation, we cannot simply blurt out a large set of candidate entities/predicates, for the user might be overwhelmed. Thus, we face the problem of giving an intermediate ranking to candidate entities/predicates, diminishing the magnitude of the pool of potential choices presented to the individual (in a top-k fashion).

Predicate Selection

A naive approach to populate the set of candidate predicates would be to scan all possible predicates and discard those whose descriptions do not match with the specified relation keywords R_{ij}. It is also possible to create a full-text index on the predicate descriptions and to query it using R_{ij}. As can be expected, this approach is too simple and yields a large set of candidate predicates.

The choice of candidate predicates p_{ij} can be improved by using the category keywords C_i, C_j and the selector keywords K_i, K_j for the entity variables x_i and x_j that are related by p_{ij}. Following the success

of generative models in entity search [14], we suggest a creative linguistic framework for collaborative query comprehension to select potential predicates.

Let $\vec{q}$ = (R_{ij}, C_i, C_j, K_i, K_j) denote the part of the query pertaining to the predicate p_{ij} and entities x_i, x_j. We are interested in $\text{argmax}_p \Pr(p|\vec{q})$, where

Latent variables t_i, t_j are introduced in (8), representing the types of the entities x_i and x_j, respectively. The expressions (I) and (II) can be simplified using the following reasonable independence assumptions:

1. $\Pr(t_i, t_j | R_{ij}, p) = \Pr(t_i, t_j | p)$, i.e., the relation keywords R_{ij} do not convey any extra information when the predicate p is known.
2. $\Pr(C_i, C_j | t_i, t_j, R_{ij}, p) = \Pr(C_i, C_j | t_i, t_j)$, i.e., the category keywords are independent of R_{ij} and p, once t_i and t_j are known.
3. $\Pr(C_i, C_j | t_i, t_j) = \Pr(C_i|t_i) \cdot \Pr(C_j | t_j)$, i.e., C_i and C_j are independent of each other and of any types other than t_i and t_j, respectively.

Furthermore, by introducing the latent variables e_i and e_j for entities represented by x_i and x_j in the query, we can rewrite ϕ_1 as:

Similarly, as before, we need the following independence assumptions to simplify (IV) and (V),

1. $\Pr(e_i, e_j | C_i, C_j, t_i, t_j, R_{ij}, p) = \Pr(e_j, e_j | t_i, t_j, p)$, i.e, the category keywords C_i, C_j, and the relation keywords R_{ij} do not convey any extra information when the types t_i, t_j, and the predicate p is known.
2. $\Pr(K_i, K_j | e_i, e_j, C_i, C_j, t_i, t_j, R_{ij}, p) = \Pr(K_i, K_j | e_i, e_j)$, i.e., the selector keywords K_i, K_j are independent of category keywords C_i, C_j, the types t_i, t_j, the relation keywords R_{ij}, and the predicate p, once the actual entities e_i, e_j are known.
3. $\Pr(K_i, K_j | e_i, e_j) = \Pr(K_i|e_i) \cdot \Pr(K_j | e_j)$, i.e., K_i and K_j are independent of each other and of any entities other than e_i and e_j, respectively.

Therefore, (14) reduces to $\Sigma_{ei,ej} \Pr(e_i, e_j | t_i, t_j, p) \cdot \Pr(K_i|e_i) \cdot \Pr(K_j | e_j)$ and (12) now becomes:

Arguments to functions ϕ_1 and ϕ_2 are omitted in Equations 1–3 and 7 for the lack of horizontal space.

Throughout this section, we have dealt with probabilities and have ultimately arrived at a set of probabilities to be computed. We understand that such probabilities in the Bayesian sense of the word are hard to estimate, if at all well-defined. We, therefore, tweak our notation a little, replacing probabilities Pr with potentials ψ. The entire derivation continues to hold, even with the modified semantics.

In the subsequent portion, we will employ the below mentioned symbolism:

1. $\psi(p)$ in place of $\Pr(p)$: represents the prior weight on the predicate p
2. $\psi(R_{ij}, p)$ in place of $\Pr(R_{ij} | p)$: represents the compatibility of keywords R_{ij} and predicate p
3. $\psi(C_i, t_i)$ in place of $\Pr(C_i|t_i)$: represents the compatibility of keywords C_i and a Freebase type t_i
4. $\psi(K_i, e_i)$ in place of $\Pr(K_i|e_i)$: represents the compatibility of keywords K_i and the Freebase entity e_i
5. $\psi(t_i, p, t_j)$ in place of $\Pr(t_i, t_j | p)$: represents the compatibility of the predicate p with the Freebase types t_i and t_j
6. $\psi(e_i, e_j, t_i, t_j, p)$ in place of $\Pr(e_i, e_j | t_i, t_j, p)$: represents the compatibility of Freebase entities e_i and e_j with Freebase types t_i and t_j, respectively, and of the entities with the Freebase predicate p

The potentials ψ above can be compactly represented as an undirected probabilistic graphical model. We omit the discussion in this direction for brevity and describe how the above quantities will be estimated.

Estimating ψ(p): ψ(p) is simply the prior on the predicate *p*. We estimate it as:

$$\psi(\mathrm{p}) = \frac{freq(p)}{\Sigma_{pi \in \mathbb{P}} freq(p_i)}$$

where $\mathbb{P}$ is the set of all possible predicates in our RDF data source.

Estimating ψ(Rij, p), ψ(Ci, ti), and ψ(Ki, ei): The potentials $\psi(R_{ij}, p)$, $\psi(C_i, t_i)$, and $\psi(K_i, e_i)$ are similar in the sense that they all capture the agreement between two variables: one of them being keywords supplied in the user query and the other being a predicate, type, or an entity in the structured data source. For example, if the relation keywords R_{ij} are "death reason" and the related entity sought is of type person, then R_{ij} should match to the predicate /people/ deceased_person/ cause_of_death rather than the predicate /people/ deceased_person/ place_of_death.

Significant diversity can arise in the representation of keywords within a query. The relational linguistic framework needs to create a link between the structured variable p and the written symbol R_{ij}, guaranteeing that improbable p values possess considerable potential. Numerous methodologies [15–18] to this problem have been studied recently. We choose a pattern-based approach akin to PATTY [19] for its simplicity and scalability to large amounts of data.

We detect the most closely linked phrase structures from a designated collection of texts for every Freebase predicate p, and afterward, these patterns are annotated within a considerably larger payload corpus. The ClueWeb09 corpus annotated with Freebase entities [20] may be used for this purpose. For each Freebase predicate p, we consider all triples and locate all the corpus sentences that mention the triple's participating entities. Taking a simplistic approach, we presume that the presence of both entities e_i and e_j in a sentence indicates support for the triple $\langle e_i, p, e_j \rangle$. Subsequently, we analyze each sentence by parsing it into a dependency graph via the Malt Parser [21].

We present the approach towards estimating $\psi(R_{ij}, p)$; those for $\psi(C_i, t_i)$ and $\psi(K_i, e_i)$ will be similar and shall be added in later versions of this work. The following brief description is based on [22].

Words along the path linking the entities are combined and incorporated into a potential phrase dictionary because the path consists of no more than k-hops (k is chosen experimentally; we expect longer dependency paths to arise out of noisy sentences/parsing). Thus, for all predicates p, we have a list $\mathbb{R}_{ij}$ of all phrases known to hint at p. We can then estimate $\psi(R_{ij}, p)$ as:

$$\psi(R_{ij}, p) = \frac{n(p, R_{ij})}{\Sigma_{R'_{ij} \in \mathbb{R}_{ij}} n(p, R'_{ij})} \quad (1)$$

where $n(p, R_{ij})$ represents the number of sentences in which R_{ij} occurred as a phrase in the dependency path between entities participating in relation specified by predicate p.

Estimating ψ(ti, tj, p)

The potential does not involve any query keywords, so it is relatively straightforward to estimate. For the predicate p, we identify all the triples it is involved in and filter those out in which the connected subject and object have types t_i and t_j, respectively. Let $n(p; t_i, t_j)$ be the number of triples. Then, simply

$$\psi(t_i, t_j, p) = \frac{n(p; t_i, t_j)}{n(p)} \quad (2)$$

where n(p) denotes the number of triples in Freebase that have p as the predicate.

Note that the filtering operation above can be easily carried out in the case of Freebase due to the availability of a /object/type/ relation for most entities.

Estimating ψ (e$_i$, e$_j$, t$_i$, t$_j$, p)

Though ψ (e$_i$, e$_j$, t$_i$, t$_j$, p) must take into account the e$_i$–t$_i$, and e$_j$–t$_j$ compatibility, we make a simplifying assumption without deviating a lot from the intended semantics of ψ(e$_i$, e$_j$, t$_i$, t$_j$, p).

Define,

$$\psi(e_i, e_j, t_i, t_j, p) = \frac{n\,(e_i, e_j; p)}{n\,(p)} \tag{3}$$

where n(e$_i$, e$_j$; p) = 1 if the triple ⟨e$_i$, p, e$_j$⟩ occurs in the data, otherwise n(e$_i$, e$_j$; p) = 0. As earlier, n(p) denotes the number of triples with p as their predicate part.

Computation

Now that we have defined the quantities involved in Equation 4, we can compute the summation involved on the right-hand side. Note that only ψ(e$_i$, e$_j$ |t$_i$, t$_j$, p) involves all the summation variables t$_i$, t$_i$, e$_i$, e$_i$ and can be expensive to compute. The summation of the other terms can be computed rather cheaply.

One way of reducing the computation costs is to plainly set ψ(e$_i$, e$_j$ |t$_i$, t$_j$, p) = 1. The burden of producing a valid score rests divided on all the ψ's involved, and we expect that the results will not be perturbed much on ignoring one of the ψ's.

Also, the arg max computation we started initially with can be replaced by a score computation for each predicate. On cleaning our Freebase dataset, we ended up with only 6,200 distinct predicates, so the score computation will be generally feasible, given that several quantities can be precomputed and stored.

Entity Selection

Similar to our approach to Predicate Selection, we use a generative language model to narrow down candidate entities for the Entity Selection stage.

Let e$_i$ denote a candidate entity (unknown) for the entity variable x$_i$ and let $\vec{q}_{ei} = (K_i, C_i)$ be part of the query pertaining directly to x$_i$. As before, we are interested in argmax$_e$Pr(e|$\vec{q}_{ei}$), where

The summation of (18) is obtained by the introduction of latent variables p$_{ij1}$, p$_{ij2}$, . . . , p$_{ijr}$, that denote the various predicates that relate xi to any other x$_j$(i≠ j) in the query. Note that we arrive at the entity selection stage only after one or more stages of predicate selection. Thus, the variables p$_{ij1}$, p$_{ij2}$, . . ., p$_{ijr}$ range over $\mathbb{P}_{ij1}, \mathbb{P}_{ij2}, \ldots, \mathbb{P}_{ijr}$, respectively, where $\mathbb{P}_{ijk}$ (1 ≤ k ≤ r) denote the set of selected candidate predicates in previous predicate selection stage(s). $\mathbb{P}_{ijk} = \emptyset$, in case predicate selection stage has not occurred for p$_{ijk}$, yet.

As previously, we make several reasonable independence assumptions to simplify (VI) and (VII),

1. Pr(C$_i$|K$_i$, e) = Pr(C$_i$|e), i.e., given the entity e, the category keywords C$_i$ are independent of the selector keywords K$_i$.
2. Pr(p$_{ij1}$, p$_{ij2}$, . . . , p$_{ijr}$ |C$_i$, K$_i$, e) =$\prod_{k=1}^{r}$ Pr(p_{ijk}|e), i.e., the predicates are independent of each other and C$_i$, K$_i$, once the entity e is known.

By now, Equation 5 has been reduced to:

$$Pr(e, \vec{q}_{ei}) = \Pr(e) \cdot Pr(K_i|e) \cdot\ Pr(C_i|e) \cdot \sum_{p_{ij1}, p_{ij2}, \ldots, p_{ijr}} \prod_{k=1}^{r} Pr\left(p_{ijk|e}\right) \tag{4}$$

As previously done for the case of entity selection, we change the semantics of the above equation and introduce ψ potential functions instead of probability mass function Pr. We now proceed towards computing the various ψ's involved.

Estimating ψ(e)

ψ(e) is simply the prior on entity e. We estimate it as follows:

$$\psi(\mathrm{e}) \approx \frac{freq(e)}{\sum_{e_i \in \mathbb{E}} freq(e_i)}$$

where $\mathbb{E}$ is the entities considered for ranking.

Estimating ψ(K_i, e)

An initial approach is to use Word-Net synsets of the short description available for the entity e; and set ψ(K_i, e) for any matching words in the synsets. More sophisticated measures can be defined with the availability of entity-annotated corpora and on-the-fly entity annotators [23].

Estimating ψ(C_i, e)

A Freebase entity can generally be associated with several types $t_1, \ldots, t_m$. We define,

$$\psi\,(C_i, e) = \sum_{j=1}^{m} \psi\big(C_i, t_j\big) \tag{5}$$

where ψ(C_i, t_j) can be used as discussed for the predicate in the previous section.

Estimating ψ(p_{ijk}, e)

It is natural to have the following definition,

$$\psi\big(p_{ijk}, e\big) = \frac{n(p_{ijk}, e)}{n(e)} \tag{6}$$

where n(p_{ijk}, e) is the number of triples that have e related by the predicate p_{ijk}, and n(e) is the total number of triples of e.

Computation

The summation in Equation 4, when rewritten in product-of-sum form, is not expected to be computationally expensive owing to the following observations:

1. The predicates $p_{ij1}, p_{ijr}, \ldots, p_{ijr}$ are the predicates that concern the entity variable ei in the query. Therefore, for even the most convoluted (and meaningful) queries one can think of, r would be a small number (< 10 let's say).
2. The predicate selection step would have been completed for these predicates before entity selection is done for the entity variable e. This constrains the entirety of items in the aggregate, enabling economic calculation of the comprehensive outcome score.

Further success can be had by precomputing some of the quantities ψ.

The arg max computation can be done away with, and top-k ranking entities can be chosen as the candidates to be presented to the user. Even the score is not required to be computed for all entities, and this step can be assisted by using a heuristic to filter a larger set of candidate entities and then computing the ranking for these entities. An initial heuristic would be to perform a keyword search using the selector keywords (and their synonyms) of the entity and to perform ranking for the result entities. Systems like Apache Solr can support a keyword search cute smiley 2015 apache.

RESULT OF THE QUERY

At this stage, we have a set of selected entities for each entity variable sought by the query and also a set of selected predicates for each of the relations between pairs of entity variables in the query. We can further assume with some confidence that the sizes of these sets would not be too large. In such a situation, arriving at the result set is a matter of computing the join of selected entity sets corresponding to adjacent entities in the query graph. Further filtering may be required to discard (entity-predicate-entity) tuples that do not have matching triples in the data.

IMPLEMENTATION

In the last section, we introduced the Entity Relationship Keyword Querying model. In the present section, we shall describe the implementation of the same. There are several possible ways to carry out many of the computations involved in query processing. The following exposition shall discuss the relative merits and demerits of the possible approaches. Most of the ideas are uncomplicated, but as is often said, the devil is in the details.

Predicate Selection

In the preceding "Query Processing: Predicate selection", it was demonstrated that a crucial aspect of query processing involves generating an ordered roster of potential predicates for every collection of relational terms R_{ij} within the query. We discussed in "Query Processing: Predicate selection" the joint interpretation ranking approach, which takes into account all parts of the query holistically to produce a ranking of candidate predicates for given relation keywords R_{ij}.

Recall that the score given to a candidate predicate is based on the following computation:

Let us assume that we have a function $\psi R(R_{ij})$: $R \rightarrow \mathbb{R}^{|\mathbb{P}|}$, which produces a score vector of length $|\mathbb{P}|$given a bag of relation keywords R_{ij}. In the above notation R denotes the set of all possible bags of relation keywords, $\mathbb{P}$ is the set of all freebase predicates, and $\mathbb{R}$ is the set of real numbers. Furthermore, let $\overrightarrow{\psi_{freq}} \in \mathbb{R}^{|\mathbb{P}|}$ denote the vector of all $\psi(p)$.

Then, we compute the dot product $\overrightarrow{\psi_{freq}} \cdot \psi_R(R_{ij}) = \overrightarrow{\psi_{local}(R_{ij})}$ that captures the compatibility of various freebase predicates with the relation keywords R_{ij}, considering R_{ij} part of the query in isolation (ignoring the existence of various category keywords C_i and selector keywords K_i etc.).

Since the summation in Equation 6 is over many 4-tuples (t_i, t_j, e_i, e_j), the computation is likely to be infeasible. It is worth noting that we need not compute the score in Equation 6 for all freebase predicates, but only for those which have high representation in the vector $\overrightarrow{\psi_{local}(R_{ij})}$, as we seek only the top-k scoring freebase predicates. Therefore, the score $\text{Score}(p|\vec{q})$ is computed for only the top-k scoring freebase predicates as per $\overrightarrow{\psi_{local}(R_{ij})}$.

Computing ψ(p)

A single pass on the freebase dump is sufficient to compute the frequencies of freebase predicates. The restricted freebase subset used for our experiments consists of 6,106 predicates, and the distribution is shown in Figure 5. The sum of all frequencies equals the number of triples in our restricted subset and is found to be 827,615,838.

For our scoring, we have the following straightforward computation:

$$\psi(p) = \frac{freq(p)}{827{,}615{,}838}$$

The various $\psi(p)$ values are collected to form the vector $\overrightarrow{\psi_{freq}}$

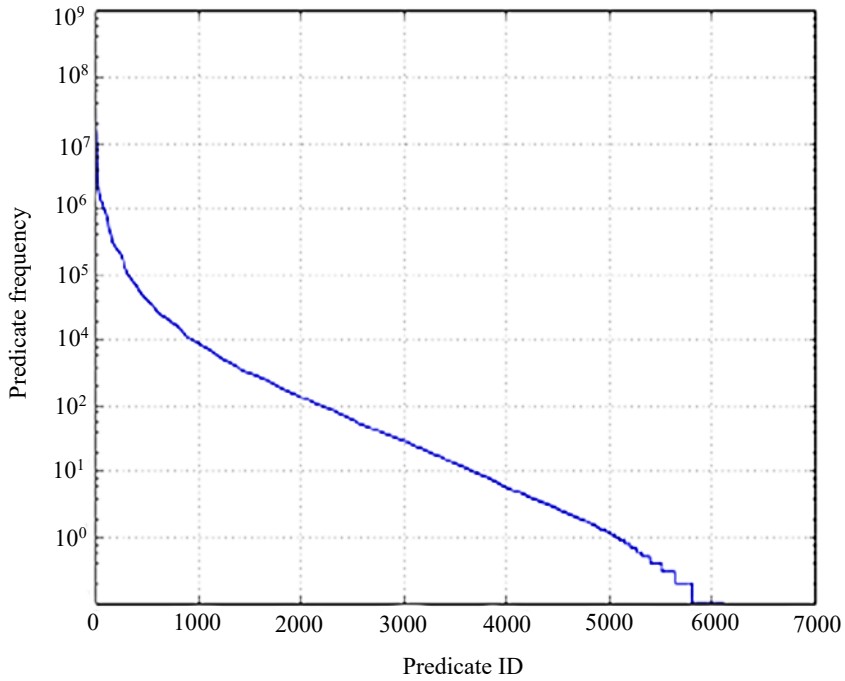

Figure 5. Frequency distribution of Freebase predicates. Note that the scale on the Y-axis is logarithmic.

Computing $\psi(R_{ij}, p)$

Recall that the potential $\psi(R_{ij}, p)$ denotes the compatibility between the relation keywords R_{ij} and the freebase predicates p. The following text briefly describes several interesting approaches we investigated for computing $\psi(R_{ij}, p)$. The last subsection describes the approach followed in [?], using a text corpus and the freebase dump.

Using Lucene-based Search

Apache Lucene [24] is a popular information retrieval library that supports full text indexing and searching capabilities. Lucene has been successfully used in the retrieval of text documents, and it is straightforward to map our problem to the text document search framework.

We construct micro-documents for each of the 6,106 freebase predicates with the following contents:

1. Name of the freebase predicate. For example, for the freebase predicate/people/person/ employment history, the phrase "employment history" is retained in the micro-document.
2. Synonyms of the freebase predicate name. WordNet may be used to construct the list of synonyms.

Lucene implements a variant of the TfIdf scoring model. The Lucene scores are returned along with the top-k search results–which may be directly used as a proxy for $\psi(R_{ij}, p)$

Using the ClueWeb09 Corpus

A method reminiscent of the one outlined in [22] holds particular relevance to our current challenge. Applying the pattern-centric methodology to each freebase predicate p independently allows for discovering the most closely linked phrase structures from a designated corpus. Subsequently, these structures are merged into a more extensive text data collection.

Our implementation entails incorporating the 6,106 freebase predicates from the restricted dump alongside the ClueWeb09 corpus annotated with freebase entities [25]. We detect all corpus sentences

referring to both participating entities for each instance of every predicate triple. Our assumption is that if $< e_1, p, e_2 >$ represents a triple and e_1, e_2 appear together in a sentence, then that sentence provides evidence of the predicate relationship. Subsequently, each sentence undergoes parsing to derive a dependency graph using the Malt Parser [26]. Terms found on the route that links the entities are combined and included in a potential phrase lexicon if the route encompasses a maximum of three transitions. Interestingly, extended dependency routes typically arise from imprecise sentences or parsing errors. We retained 30% of the sentences. Finally, we compute the potential $\psi(R_{ij}, p)$ as follows:

$$\psi(R_{ij}, p) = \frac{n(p, R_{ij})}{\sum_{R'} n(p, R')}$$

In this context, R' spans all expressions recognized as signifying p, while n(p, R') signifies the count of instances where the expression R' was present within the path connecting the entities associated with relation p.

Although this model operates under simplistic assumptions, such as equating entity co-occurrence with evidence, its primary objective is to identify the top-k freebase predicates that align with the relation keywords R_{ij}. The collective scoring mechanism aids in mitigating residual noise, as per Equation 6. With guidance from the authors of [22], we were able to repurpose a significant portion of the CSAW(CSAW: Curating and Searching the Annotated Web. URL: www.cse.iitb.ac.in/~soumen/ doc/CSAW/) codebase, tailored to our specific requirements.

A relation model relying exclusively on corpus annotations might face challenges stemming from either sparse annotations or the infrequent occurrence of Freebase triples within the ClueWeb dataset. To illustrate, the Freebase relation /people/ person/profession contains a limited number of annotated sentences. Consequently, we incorporate Freebase relation type names into a language model as lemmas to mitigate this issue. For instance, profession contributes to the relation type/people/person/profession.

To conclude this subsection, we present a sample of the top-k outcomes obtained using the ClueWeb09 approach.

Concluding Remarks: Even though the pattern-based approach utilizing ClueWeb09 corpus seems superior to the naive Lucene-based search technique, in practice, we observed the latter to be much lighter on resources (and, thus, faster) without compromising a lot on the accuracy of scores.

Computing $\psi(C_i, t_i)$

Similar to the $\psi(R_{ij}, p)$ computation discussed in the previous section, the Lucene-based search approach can be adapted to the $\psi(C_i, t_i)$ computation capturing the similarity between a bag of category keywords Ci and a freebase type t_i.

Determining the entity type without being too broad or too specific is vital for effective Question-Answering (QA). In an example scenario where Ci represents "city," an ideal type language model would favor t_i as /location/citytown over/location/location, while also steering clear of /location/es_ autonomous_city.

One straightforward strategy for constructing a type language model involves leveraging catalogs like Freebase. Each Freebase type is defined by a set of expressions obtainable via the /common/topic/ alias link. These phrases could be aggregated into a compact text and utilized to create a uniform Dirichlet-smoothed linguistic framework in accordance with concepts from Information Retrieval (IR) [27]. Furthermore, in Freebase, an entity node (e.g., Einstein, /m/0jcx) can connect to a type node (e.g., /base/scientist/physicist) through an edge labeled /type/object/type. Moreover, connection categories within Freebase offer additional perspectives on the classifications of terminal objects. For instance, the (directed) connection category /location/country/capital establishes a connection from /location/country

to /location/citytown, suggesting that "capital" might belong to the array of defining terms for the object classification /location/citytown.

It is noteworthy that while Freebase link naming conventions are employed for relation and type language models, these models are also adaptable to other catalogs. Catalogs typically employ established methods to generate language models describing their structures; as an example, YAGO categories originate from WordNet groupings with linked sentence explanations, whereas YAGO connections showcase clear titles such as actedIn and isMarriedTo, providing valuable data for estimating language models. Similarly, DBPedia relations primarily stem from meaningfully named attributes sourced from infoboxes, making them directly usable. Furthermore, other research efforts [28] have demonstrated techniques for associating language models with such relations.

The type language model discussed above was accessible within the CSAW codebase and was utilized with slight adjustments.

Below are two example executions of the aforementioned approach, each yielding the top-k Freebase types (accompanied by scores) based on a collection of category keywords C_i.

Computing $\psi(K_i, e_i)$

The Lucene-based search approach is better suited for computing $\psi(K_i, e_i)$, than it is for computing $\psi(C_i, t_i)$ and $\psi(R_{ij}, p)$. This is true mainly for two reasons:

1. The micro-document for an entity e_i can utilize the freebase description available for that entity. Most freebase entities are linked to their description via the predicate/common/topic/description. In general, the description is available in a number of languages; we restrict ourselves only to the English-language description available for entities.
2. The availability of alias links (via the freebase predicate/common/topic/alias for entities) also aids the microdocument generation.

Furthermore, the word embeddings approach for estimating $\psi(K_i, e_i)$ is straightforward too since Google has made available (Google Code Archive - word2vec. URL: code.google.com/archive/p/word2 vec/) pre-trained entity vectors with freebase naming. The word2vec 'distance' function can be called to return the freebase entities with the least cosine distance from the keyword query K_i, and the cosine distance itself can be used in lieu of $\psi(K_i, e_i)$.

Computing $\psi(t_i, t_j, p)$

Recall that we defined

$$\psi(t_i, t_j, p) = \frac{n(p;t_i,t_j)}{n(p)}$$

where $n(p; t_i, t_j)$ denotes the number of triples in freebase that contain the predicate p and whose subject and object have the freebase types t_i and t_j, respectively. n(p) as before, denotes the number of freebase triples containing p as the predicate.

To compute $n(p; t_i, t_j)$, we iterate over the freebase triples dump and create a new intermediate dump in which each of the subject and object are replaced by their respective freebase types. To speed up this replacement pass, we were required to maintain an in-memory dictionary mapping each freebase entity to its freebase type. If the item within the trio wasn't identified as a freebase URL, we just disregarded the trio altogether.

Once the intermediate dump was produced, an external merge-sort of the dump was performed to cluster the rows with the same value of (t_i, t_j, p). A final pass computed the quantity n(p; ti, tj) by counting the number of adjacent rows with the same value of (t_i, t_j, p).

The final n(p; t_i, t_j) values are stored in a PostgreSQL table, along with n(p) values in another table. The two quantities are retrieved from PostgreSQL to compute ψ(t_i, t_j, p) when required during query processing.

Overall score computation for predicate selection

Among the quantities in 21, we have so far discussed the computation of ψ(p), ψ(R_{ij}, p), ψ(C_i, t_i), ψ(K_i, e_i) and ψ(t_i, t_j, p). Since the summation in Equation 6 is overall t_i, t_j, e_i, e_j involved, the exact computation will likely be infeasible within reasonable expectations of query processing time. We therefore, for our prototyping purposes, drop the summation over e_i, e_j by simply setting ψ(e_i, e_j, t_i, t_j, p) = 1, ψ(K_i, e_i) = 1, and ψ(K_j, e_j).

Our simplified scoring model for predicates is, thus,

$$Score(p|\vec{q}) \propto \psi(\mathrm{p}) \cdot \psi\left(R_{ij}, p\right) \cdot \sum_{t_i,t_j} \psi(C_i, t_i) \cdot \left(C_j, t_j\right) \cdot \psi\left(t_i, t_j, p\right) \quad (7)$$

We use Lucene scores as a proxy for ψ(R_{ij}, p) for our prototype implementation. An Apache Solr index is built on micro-documents (one for each freebase predicate) containing WordNet synsets of the freebase predicate name. In the first step, the relation keywords R_{ij} are used to retrieve the top-100 freebase predicates and their Lucene scores by querying the Solr index. Second, the language model for types is queried by C_i and C_j to return two lists of candidate types (along with ψ(C_i, t_i) scores) of the formal types t_i and t_j.

Once we have the list of candidate predicates and two lists for candidate types and their respective scores, we probe each combination of (ti, tj, p) and fetch ψ(t_i, t_j, p) using a PostgreSQL query. The product ψ(C_i, t_i) · ψ(C_j, t_j) · ψ(t_i, t_j, p) is finally marginalized by summing over all candidate types for t_i and t_j.

The list of predicate scores thus obtained is sorted, and the top 100 predicates are sent to the user for selection. The user may select one or more predicates, concluding one predicate selection step.

Entity Selection

For our initial prototype, we replace the earlier sophisticated entity ranking and selection model with a simple ranking of entities by ψ(K_i, e_i) computed earlier.

Micro-documents for each freebase entity are constructed, each containing the freebase description of the entity (linked via the predicate /common/topic/description) and its aliases (linked via the predicate /common/topic/alias). A Solr index is created on these micro-documents. Note that this index is separate from that created for freebase predicates.

In the entity selection step for e_i, the Solr index is queried with selector keywords K_i, which returns a list of freebase entities and their corresponding Lucene scores for the query. The selected predicates (for a predicate p adjoining e_i in the query graph) from a previous predicate selection step are joined with the entities returned by Solr to form triple patterns of the form <e_i, p, ?o>. A freebase dump set up on Openlink Virtuoso is queried with these triple patterns using Virtuoso's SPARQL endpoint. The set of entities for which any of the corresponding SPARQL queries return non-empty results are retained. These retained entities are ranked by the Lucene scores (already found from the initial Solr query) and presented to the user for selection.

QUERY RESULT

Once the user has performed predicate selection for all unknown predicates and entity selection for all unknown entities in the query, it is straightforward to join the sets of selected entities and predicates and return a result graph. Efforts are made to guarantee that every boundary of the resultant graph is documented as triads within the freebase dump.

The result graphs are ranked by a simple score computation: The evaluation of a graph is determined by multiplying the evaluations of every proposition and object within the graph. The scores computed during predicate and entity selection are used for the result graphs in this final score computation.

CONCLUSION AND FUTURE WORK

The Entity Relationship Keyword Querying (ERKQ) (Henceforth shall be referred to as ERKQ in all citations) framework provides a powerful combination of ease of use and the capability of processing complicated queries beyond the scope of existing systems. The former feature stems from the fundamental decision of permitting the user to query using free keywords, albeit expecting it in a pre-segmented form. This pre-segmentation and a carefully crafted query interpretation layer give the framework the latter feature [29].

Staying within the ERKQ model, multiple additions are possible in the query processing mechanism that can improve the system's overall performance.

Recall that the query processing mechanism was described as involving the repetition of two actions (predicate selection and entity selection) for various elements (predicates and entities) of the query. However, the order of actions that need to be taken is not clear. We believe that great improvements are possible with work towards this end.

The idea of repeating the two selections (say alternatively) is a simple one. Two (or more) selection steps could be combined into one. Even the idea of computing the joins at the end could be extended to computing intermediate joins and prompting the user for more intermediate selections. In approaches like the latter, there is always the inherent trade-off between ease of use (or user discontent) and accuracy/precision of results.

Sophisticated potential functions need to be designed, and their effectiveness needs to be tested.

Last, we acknowledge that our prototype implementation leaves a lot to be desired. Significant implementation improvements would be necessary in order to show the efficacy of entity relationship keyword queries and our approach.

REFERENCES

1. DuCharme B. Learning SPARQL: Querying and updating with SPARQL 1.1. Sebastopol, CA, USA: O'Reilly Media, Inc.; 2013.
2. Siemer S. Exploring the Apache Jena framework. Göttingen, Germany: George August University; 2019.
3. Ryen V, Soylu A, Roman D. Building semantic knowledge graphs from (semi-) structured data: A review. Future Internet. 2022;14 (5): 129.
4. Dosso D, Silvello G. Search text to retrieve graphs: A scalable RDF keyword-based search system. IEEE Access. 2020; 8: 14089–14111.
5. Bhalotia G, Hulgeri A, Nakhe C, Chakrabarti S, Sudarshan S. Keyword searching and browsing in databases using BANKS. In: Proceedings of the 18th International Conference on Data Engineering; 2002 Feb 26-Mar 1; New York, NY, USA: IEEE; 2002. 431–440 p.
6. Nie Z, Ma Y, Shi S, Wen J-R, Ma W-Y. Web object retrieval. In: Proceedings of the 16th International Conference on World Wide Web; 2007 May 8–12; New York, NY, USA: ACM; 2007. 81–90 p.
7. Elbassuoni S, Blanco R. Keyword search over RDF graphs. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management; 2011 Oct 24-28; New York, NY, USA: ACM; 2011. 237–242 p.
8. Bakhshi M, Nematbakhsh M, Mohsenzadeh M, Rahmani AM. Data-driven construction of SPARQL queries by approximate question graph alignment in question answering over knowledge graphs. Expert Syst Appl. 2020; 146: 113205.

9. Yahya M, Berberich K, Elbassuoni S, Ramanath M, Tresp V, Weikum G. Deep answers for naturally asked questions on the web of data. In: Proceedings of the 21st International Conference Companion on World Wide Web; 2012 Apr 16-20; New York, NY, USA: ACM; 2012. 445–449 p.
10. Ao J, Chirkova R. KGIQ: Scalable translation of user-specified examples into knowledge-graph queries. In: Proceedings of the 2022 IEEE International Conference on Big Data; 2022 Dec 17–20; Osaka, Japan. New York, NY, USA: IEEE; 2022. 3909–3918 p.
11. Demidova E, Zhou X, Nejdl W. FreeQ: An interactive query interface for Freebase. In: Proceedings of the 21st International Conference Companion on World Wide Web; 2012 Apr 16-20; New York, NY, USA: ACM; 2012. 325–328 p.
12. Nasar Z, Jaffry SW, Malik MK. Named entity recognition and relation extraction: State-of-the-art. ACM Comput Surv. 2021; 54 (1): 1–39.
13. Guo J, Cai Y, Fan Y, Sun F, Zhang R, Cheng X. Semantic models for the first-stage retrieval: A comprehensive review. ACM Trans Inf Syst. 2022; 40 (4): 1–42.
14. Arsalane W. Exploring generative adversarial networks for entity search and retrieval. In: Proceedings of the International Conference on Smart Computing and Cyber Security: Strategic Foresight, Security Challenges and Innovation (SMARTCYBER 2020); 2021 Aug 23–25; Singapore. Singapore: Springer; 2021. 55–68 p.
15. Berant J, Chou A, Frostig R, Liang P. Semantic parsing on Freebase from question-answer pairs. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing; 2013 Oct 18–21; Stroudsburg, PA, USA: ACL; 2013. 1533–1544 p.
16. Berant J, Liang P. Semantic parsing via paraphrasing. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014); 2014 Jun 22–27; Baltimore, MD, USA: ACL; 2014. 92 p.
17. Kwiatkowski T, Choi E, Artzi Y, Zettlemoyer L. Scaling semantic parsers with on-the-fly ontology matching. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing; 2013 Oct 18–21; Stroudsburg, PA, USA: ACL; 2013. 1545–1556 p.
18. Yih WT, He X, Meek C. Semantic parsing for single-relation question answering. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014); 2014 Jun 22–27; Stroudsburg, PA, USA: ACL; 2014. 643–653 p.
19. Nakashole N, Weikum G, Suchanek F. PATTY: A taxonomy of relational patterns with semantic types. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL 2012); 2012 Jul 12–14; Stroudsburg, PA, USA: ACL; 2012. 1135–1145 p.
20. Winston E, Dhingra B, Mazaitis K, Neubig G, Cohen WW. Cloze question answering using semi-structured knowledge and a language model. [Conference paper]. 2020. [Online] Available at https://ezrawinston.github.io/files/case_qa.pdf [Accessed on August 2024]
21. Anderson M, Gómez-Rodríguez C. Inherent dependency displacement bias of transition-based algorithms. arXiv [Preprint]. 2020 [Online]. Available at https://arxiv.org/abs/2003.14282 [Accessed on August 2024]
22. Joshi M, Sawant U, Chakrabarti S. Knowledge graph and corpus driven segmentation and answer inference for telegraphic entity-seeking queries. [Conference paper]. 2014. [Online] Available at https://aclanthology.org/D14-1117/ [Accessed on August 2024]
23. Sageder C, Karampatakis S. Annotating entities with fine-grained types in Austrian court decisions. In: Proceedings of the SEMANTiCS 2021 Conference; 2021 Sep 6–9; New York, NY, USA: Springer; 2021. 139–153 p.
24. Liu R, Liang J, Jin P, Wang Y. MMH-index: Enhancing Apache Lucene with high-performance multi-modal indexing and searching. In: Proceedings of the 30th ACM International Conference on Multimedia; 2022 Oct 10–14; New York, NY, USA: ACM; 2022. 7279–7289 p.
25. Westen H van, Hasibi F, de Vries A. Effect of surface form dictionary on effectiveness in entity linking. [Conference paper]. 2021. [Online] Available at https://www.cs.ru.nl/bachelors-theses/2021/Hermen_van_Westen___s4797620___Effect_of_Surface_Form_Dictionary_on_Effectiveness_in_Entity_Linking.pdf [Accessed on August 2024]

26. Kesav RS, Premjith B, Soman K. Dependency parser for Hindi using integer linear programming. In: Proceedings of the 5th International Conference on Advances in Computing and Data Sciences (ICACDS 2021); 2021 Apr 23–24; New York, NY, USA: Springer; 2021. 42–51 p.
27. Zhu R, Tu X, Huang JX. Deep learning on information retrieval and its applications. In: Wu F, Weld DS, editors. Deep learning for data analytics. Amsterdam, Netherlands: Elsevier; 2020. 125–153 p.
28. Wu F, Weld DS. Autonomously semantifying Wikipedia. In: Proceedings of the 16th ACM Conference on Information and Knowledge Management; 2007 Nov 6–10; New York, NY, USA: ACM; 2007. 41–50 p.
29. Meghana G, Chavali DP. Examining the dynamics of COVID-19 misinformation: social media trends, vaccine discourse, and public sentiment. Cureus. 2023; 15 (11): e48239.

