SLQ: A User-friendly Graph Querying System

Shengqi Yang[1] Yanan Xie[2] Yinghui Wu[1] Tianyu Wu[2]
Huan Sun[1] Jian Wu[2] Xifeng Yan[1]
[1]University of California Santa Barbara [2]Zhejiang University
{sqyang, yinghui, huansun, xyan}@cs.ucsb.edu
{xyn, tywu, wujian2000}@zju.edu.cn

ABSTRACT

Querying complex graph databases such as knowledge graphs is a challenging task for non-professional users. In this demo, we present SLQ, a user-friendly graph querying system enabling schemaless and structureless graph querying, where a user need not describe queries precisely as required by most databases. SLQ system combines searching and ranking: it leverages a set of transformation functions, including abbreviation, ontology, synonym, etc., that map keywords and linkages from a query to their matches in a data graph, based on an automatically learned ranking model. To help users better understand search results at different levels of granularity, it supports effective result summarization with "drill-down" and "roll-up" operations. Better still, the architecture of SLQ is elastic for new transformation functions, query logs and user feedback, to iteratively refine the ranking model. SLQ significantly improves the usability of graph querying. This demonstration highlights (1) SLQ can automatically learn an effective ranking model, *without* assuming manually labeled training examples, (2) it can efficiently return top ranked matches over noisy, large data graphs, (3) it can summarize the query matches to help users easily access, explore and understand query results, and (4) its GUI can interact with users to help them construct queries, explore data graphs and inspect matches in a user-friendly manner.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

Keywords

schemaless graph querying; keyword query; graph databases

1. INTRODUCTION

Graph querying is widely adopted to retrieve information from emerging graph databases, *e.g.,* knowledge graphs, information and social networks. Given a query, it is to find reasonable top answers from a data graph. Searching real-life graphs, nevertheless, is not an easy task especially for non-professional users. (1) Either no standard schema is available, or schemas become too complicated for users to completely possess. (2) Graph queries are hard to write and interpret. Structured queries (*e.g.,* XQuery [4] and SPARQL [5,7]) require the expertise in complex grammars, while keyword queries [8,10] can be too ambiguous to reflect user search intent. Moreover, most of these methods adopt predefined ranking models [5,8], which is barely able to bridge the gap between queries and their desired matches. (3) Moreover, it is a daunting task for users to inspect a large number of matches produced from querying large-scale heterogeneous graph data.

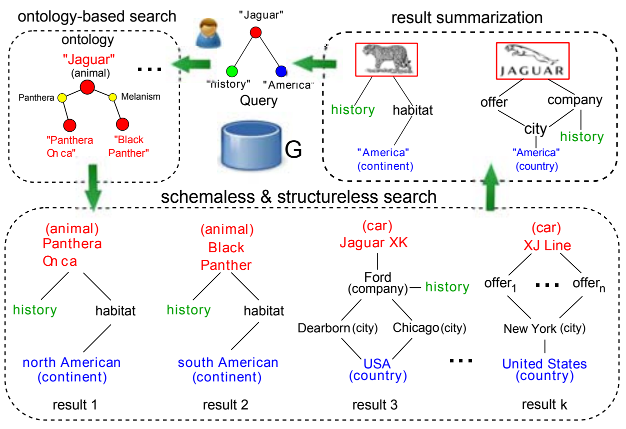


Figure 1: Searching a knowledge graph

Example 1: Consider a query asking "tell me about the history of Jaguar in America". The query can be presented as either a keyword query "Jaguar history America", or a small graph in Fig. 1. To find answers for such a simple query is, nevertheless, not easy. (1) A keyword *e.g.,* "Jaguar" may not have identical matches, but instead can be matched with entities that are semantically close, *i.e.,* luxury cars or animals. *How to find matches that are semantically related to the query?* (2) A large number of possible answers can be identified by various matching mechanisms. For example, "Panthera Onca" is closely related with "Jaguar" as its scientific name, while "Jaguar XK" is another match obtained simply by string transformations. *Which one is better?* A ranking model should be employed and tuned with or without manual tuning effort. (3) There are a large number of good results, *e.g.,* different species related to Jaguar (result

1 and result 2), or various car prototypes (result 3 to result k). *How to help users better understand results without inspecting them one by one?* This kind of complexity contrasts to the impatience of web users who are only interested in finding good answers in a short time. □

To answer these questions, we propose SLQ, a novel graph querying system for schemaless and structureless querying [13]. (1) To better understand search intent, SLQ interprets queries with external ontologies to find semantically close matches. (2) It automatically supports multiple mapping functions, namely, *transformations*, *e.g.,* synonym, abbreviation, ontology, to identify reasonable answer via learning to rank, and works with both (a) a cold-start strategy that requires no manual effort for system tuning, and (b) a warm-start strategy to adjust the ranking model with available user feedback and query logs. (3) To help users better understand results and refine queries, it supports concise result summarization. Users may inspect small summaries, and then decide to (a) drill-down for details, or (b) interactively refine queries with interesting summaries. To the best of our knowledge, these features are not seen before in any previous graph querying systems (*e.g.,* [4–8]).

SLQ system is among the first efforts of developing a unified framework for schemaless and structureless querying. Designed to help users access complex graphs in a much easier and powerful manner, it is capable of finding high-quality matches when structured query languages do not work.

Queries and datasets. SLQ supports a wide range of queries. Users may issue (1) a keyword query as a set of keywords, where each keyword describes an entity; or (2) a (not necessarily connected) graph query, where each query node has a set of labels as conditions, and an edge between two nodes, if any, specifies the relation constraints posed on two query nodes by users. We demonstrate SLQ over three major knowledge graphs, *i.e.,* DBpedia, YAGO2 and Freebase as described in the table below. A single such knowledge graph could have more than 10K types of entities, making it difficult for users to fully grasp.

Graphs	entities	relations	node types	relation types
DBpedia [1]	3.7M	20M	359	800
YAGO2 [5]	2.9M	11M	6,543	349
Freebase [2]	40.3M	180M	10,110	9,101

Table 1: Knowledge graphs

We next introduce SLQ and its features in detail.

2. ONTOLOGY-BASED QUERYING

In contrast to conventional graph querying systems, SLQ does not limit itself with single, fixed search semantic. Instead, it employs a set of matching functions that are more capable to find good matches in heterogeneous graphs [13]. We start by demonstrating *ontology-based search*, as an example for various matching mechanisms integrated in SLQ.

One of the challenges is to find the semantically related matches. SLQ leverages ontology-based search [12] to bridge the entities from queries and data graphs via a set of ontology closeness metrics. Given external ontology graphs (*e.g.,* DBPedia Ontology [1]), SLQ finds the semantically related entities (specified by a closeness measure) in the ontology graphs for each entity (keyword) in a query [12]. A straightforward "substituting-and-querying" method may next interpret the query by substituting the keywords with their related entities, which in turn have matches in the data graph. Top matches can then be extracted by processing each new query. However, it may yield a tremendous number of queries. Instead, SLQ leverages an effective ontology index [12]. In a nutshell, it computes several sketches of the data graph using the ontology graph. Each sketch is induced by grouping the nodes that are semantically close. Upon receiving a query, SLQ can efficiently identify the top matches by querying on these small sketches only.

Example 2: While there are no similarly labeled entities for "Jaguar" in the data graph (Fig. 1), SLQ checks an ontology graph, and identifies its two semantically related matches (as a type of animal): "Panthera Onca", its scientific name and "Black Panther", its melanistic color variant. These can hardly be found by conventional IR metrics or string similarity, or by using the query and data graph alone. □

SLQ wraps ontology-based searching with (a) an ontology transformation function that maps a keyword to a set of valid entities, and (b) an ontology index. These are seamlessly integrated in the query processing of SLQ (Section 3).

3. QUERYING PROCESSING MODULE

SLQ does not require a user to describe a query precisely as required by most search applications. To render this high flexibility to the users, in the core of the framework is a mechanism of *transformation*: given a query, the query evaluation is conducted by checking if its matches in a graph database can be reasonably "transformed" from the query through a set of transformation functions.

Transformations. To characterize query matches, SLQ adopts transformations defined on the attributes and values of both nodes and edges of the query [13]. A query node (or edge) has a match if it can be converted to the latter via a transformation. For example, a query node "IBM" shall be matched to a node with label "International Business Machines." The table below summarizes several common transformations supported by SLQ.

Category	Transformations
String	First token, Last token, Abbreviation, Prefix, Acronym, Drop word, Bag of words
Semantic	Ontology, Synonym (WordNet [3])
Numeric	Unit conversion, Date gap, Date conversion, Numeric range
Topology	Distance, Labeled path

Better still, SLQ exposes a generic interface abstracted for any transformation. New transformations complying with the interface can be readily plugged in.

Example 3: Recall the query in Fig. 1. SLQ captures the matches for the keyword "Jaguar" as follows. (1) The matches "Panthera Onca" and "Black Panther" can be identified by ontology transformation (Section 2). (2) The match "Jaguar XK" can be matched by "First token", a String transformation. (3) The match "XJ Line" can be captured by "Bag of words", indicating that "Jaguar" appears in its text description. For edge ("Jaguar", "America"), an topology transformation "Distance" maps it to a path from "Panthera Onca" to "north America" in result 1. □

Matching Quality Measurement. To measure the quality of the matches induced by various transformations, SLQ adopts a ranking function of *weighted transformations*. The function incorporates two types of score functions: node matching and edge matching score function. Each score function aggregates the contribution of all the possible transformations with corresponding weights. More specifically, by harnessing the probabilistic graphical model, we define the ranking function with *Conditional Random Fields* [9], since it satisfies two key considerations: using training data, its *formulation* could optimize the weights of the transformations for good ranking quality; the *inference* on the model provides a mechanism to search top-k matches quickly.

Ranking model learning. A key issue is how to select a ranking function by determining reasonable weights for the transformations. Instead of assigning equal weights or calibrating the weights by human effort, SLQ automatically *learns* the weights from a set of *training instances*. Each instance is a pair of a query and one of its relevant (labeled as "good") answers. The learning aims to identify for each transformation a weight, such that if applied, the model ranks the good answers as high as possible for each instance.

SLQ begins with a cold-start stage where no manual effort is required for instances labeling, and can be "self-trained" in the warm-start stage, using user feedback and query logs. (Cold-start) When no user query log is available, SLQ randomly extracts a set of subgraphs from the data graph as "query templates." It then injects a few transformations to the template and generates a set of training queries. Intuitively, each training query should have the corresponding templates as its "good matches," since the query can be transformed back to the template.

Query Processing. SLQ efficiently finds top matches, leveraging *approximate inferencing* [9]. It treats a query as a graphic model and the matches as assignment to its random variables (query nodes). By propagating messages among the nodes in the graphical model, the inference can identify top assignments (matches) that maximize the joint probability (ranking score) for the model. Together with a graph sketch data structure, this technique dramatically reduces the query processing time, with only small loss of match quality (less than 1% in our validation).

4. RESULT SUMMARIZATION

Due to the sheer volume of data, graph querying usually generates a lot of results that are too many to inspect. This not only makes the understanding of the results a daunting task, but also frustrates the users to continue refining the search. The result summarization feature of SLQ addresses the two challenges in a "two-birds-with-one-stone" way by leveraging [11]. (1) Given a query and a set of matches, SLQ effectively describes all the matches (as graphs) with a few *summary graphs*. A summary graph preserves the connectivity of the keyword pairs. By reviewing the summary graphs, users easily get intuitive "big pictures" of all the matches. (2) The summary graphs can further be used to refine the search by suggesting new query nodes or edges, or be issued directly as new queries.

Example 4: SLQ provides two intuitive summaries for the matches in Fig. 1. The first summary indicates that "Jaguar" refers to animals living in America continents with evolution history. The second shows that it refers to a certain type of cars sold by dealers and companies in major cities of USA, with the company history in car industry. In addition, new nodes or keywords, *e.g.,* "offer", are suggested to users to inspire queries with new interests. □

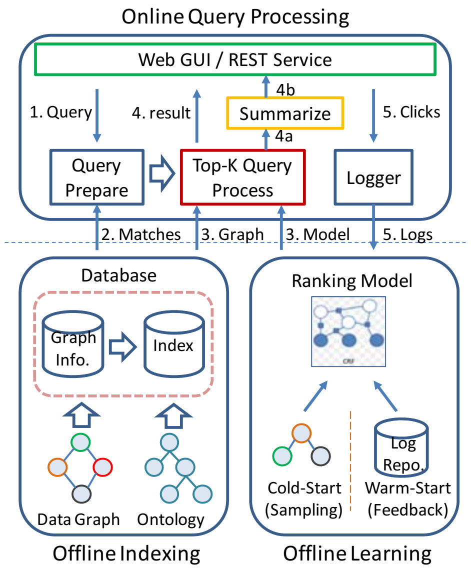


Figure 2: SLQ: Architecture

5. SYSTEM DESIGN

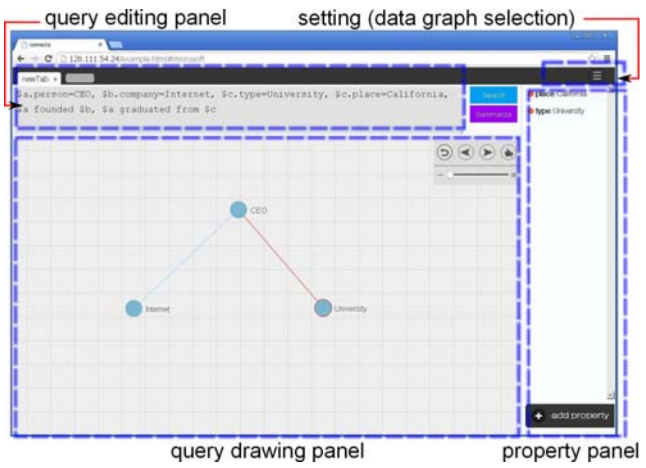
The system SLQ consists of back-end and front-end modules, as illustrated in Fig. 2. For the back-end, SLQ integrates (1) a full fledged indexing module (the lower left part in Fig. 2) which implements all the indices in Section 3, and (2) an offline learning module for the ranking model (the lower right part in Fig. 2). Each module in the back-end can be maintained dynamically in response to data updates, without affecting the front-end, *i.e.,* online query processing. We also separate the information for graph structure and its node and edge content, and store the latter in the database. This enables SLQ to perform fast graph traversal by only loading much smaller graph structure into the memory, while accessing richer content, *e.g.,* attribute values, documents, pictures in the database.

The front-end modules reside in the application layer (online query processing), as shown in the upper level in Fig. 2. Upon receiving a query, (1) the "`query prepare`" module first interprets the query to an internal format, and prepares the match candidates by looking up the indices; (2) SLQ next invokes "`top-k query process`" using the ranking model to find accurate matches; (3) once the top-k results are generated, SLQ directly renders the results to the user through our "`GUI/REST service`"; (4) SLQ can also provide the users with the summarized views of the results via "`summarize`"; and (5) the user queries and result preferences are memorized by "`Logger`" to improve the ranking model.

SLQ is designed with elasticity. It can be scaled up easily by duplicating a module without affecting the others. For example, when there are intensive query requests, we can duplicate the application layer in multiple servers and then evenly distribute the queries among the servers.

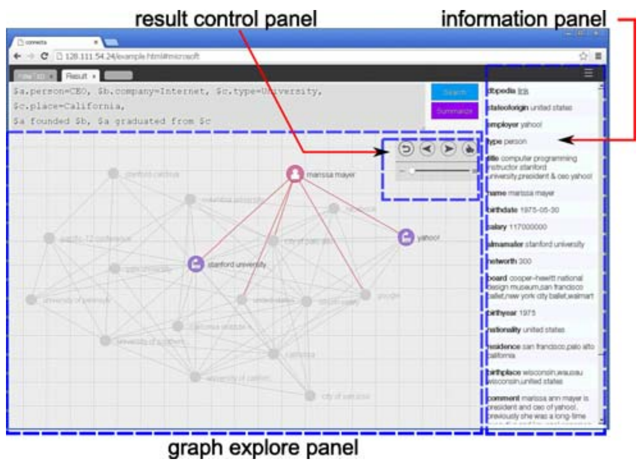
6. DEMONSTRATION OVERVIEW

Setup. We will demonstrate SLQ over the knowledge graphs in Table 1. The system is implemented in Java and is deployed on an Intel Core i7 2.8GHz, 32GB server.



```
$a.person=CEO, $b.company=Internet, $c.type=University, $c.place=California,
$a founded $b, $a graduated from $c
```

(a) Query formalization interface



```
$a.person=CEO, $b.company=Internet, $c.type=University,
$c.place=California,
$a founded $b, $a graduated from $c
```

dbpedia link
stateoforigin united states
employer yahoo!
type person
title computer programming instructor stanford university,president & ceo yahoo!
name marissa mayer
birthdate 1975-05-30
salary 117000000
almamater stanford university
networth 300
board cooper-hewitt national design museum,san francisco ballet,new york city ballet,walmart
birthyear 1975
nationality united states
residence san francisco,palo alto california
birthplace wisconsin,wausau wisconsin,united states
comment marissa ann mayer is president and ceo of yahoo!. previously she was a long-time

(b) Graph exploration interface

Figure 3: SLQ: User interface

Demo Scenario. SLQ provides user interfaces for *query formalization* and *query result exploration* (Fig. 3). We invite users to experience (1) how to easily form a query without prior training in query languages or graph databases, (2) how the results, summaries and even the data graph can be conveniently navigated and viewed.

As shown in Fig 3(a), the query formalization interface renders the users with two major *query panels* to form queries. (1) Users can conveniently draw a graph query in the `query drawing panel`, and can freely add and edit the properties and values in the the `property panel` for each query node or edge. (2) Alternatively, users are invited to use our built-in query language SLQL in the `query editing panel`. SLQL is designed for simplicity. It consists of two types of statements: (1) the *node statement*, "$\$x.property = value$", where "$\$x$" denotes a query node with a label constraint "$property = value$", and (2) the *edge statement*, "$\$x$ *predicate* $\$y$", where "$\$x$" and "$\$y$" are the query nodes as in (1) and "*predicate*" indicates the relationship between the two nodes. A graph query and its SLQL representation is shown in Fig. 3(a).

The second demonstration scenario invites users to run their queries and inspect the results. A user can also run the query on various knowledge graphs, *e.g.,*Freebase, chosen from `setting`. The results of the query in Fig. 3(a) are shown in the graph exploration interface (Fig 3(b)). The `graph explore panel` renders top-1 result (the highlighted part) as well as its peripheral structure (the dim part) in the data graph. The user can then inspect the detail information of a node/edge shown in the `information panel`. By double-clicking a node, users are able to explore the one-hop neighbors of the node from the data graph. Moreover, the `result control panel` enables the user to navigate the next/previous result or return to the top-1 result. The user feedback will be recorded if they click the `like button` on specific results. We also invite users to try the `summarize` button to view the summarized results from a large collection of returned matches, and drill down to find details.

7. ACKNOWLEDGMENTS

This research was sponsored in part by the Army Research Laboratory under cooperative agreements W911NF-09-2-0053, NSF IIS-0954125 and NSF IIS-0905084. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.

8. REFERENCES

[1] Dbpedia. *dbpedia.org*.
[2] Freebase. *freebase.com*.
[3] Wordnet. *wordnet.princeton.edu*.
[4] D. Chamberlin et al. XQuery 1.0: An XML Query Language. W3C Working Draft, June 2001.
[5] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, and G. Weikum. Yago2: Exploring and querying world knowledge in time, space, context, and many languages. In *WWW*, 2011.
[6] D. Mottin, M. Lissandrini, V. Yannis, and T. Palpanas. Exemplar queries: Give me an example of what you need. In *VLDB*, 2014.
[7] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *TODS*, 34(3), 2009.
[8] J. Pound, I. F. Ilyas, and G. Weddell. Expressive and flexible access to web-extracted data: a keyword-based structured query language. In *SIGMOD*, 2010.
[9] C. Sutton and A. McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, 93:142–146, 2007.
[10] H. Wang and C. Aggarwal. A survey of algorithms for keyword search on graph data. *Managing and Mining Graph Data*, pages 249–273, 2010.
[11] Y. Wu, S. Yang, M. Srivatsa, A. Iyengar, and X. Yan. Summarizing answer graphs induced by keyword queries. *VLDB*, 6(14), 2013.
[12] Y. Wu, S. Yang, and X. Yan. Ontology-based subgraph querying. In *ICDE*, 2013.
[13] S. Yang, Y. Wu, H. Sun, and X. Yan. Schemaless and structureless graph querying. *VLDB*, 7(7), 2014.