Querying Knowledge Graphs by Example Entity Tuples (Extended Abstract)

Nandish Jayaram[1] Arijit Khan[2] Chengkai Li[1] Xifeng Yan[3] Ramez Elmasri[1]

[1]*University of Texas at Arlington,* [2]*Nanyang Technological University, Singapore,* [3]*University of California, Santa Barbara*

***Abstract*—We witness an unprecedented proliferation of knowledge graphs that record millions of entities and their relationships. While knowledge graphs are structure-flexible and content-rich, they are difficult to use. The challenge lies in the gap between their overwhelming complexity and the limited database knowledge of non-professional users. As an initial step toward improving the usability of knowledge graphs, we propose to query such data by example entity tuples, without requiring users to form complex graph queries. Our system, GQBE (Graph Query By Example), automatically discovers a weighted hidden maximum query graph based on input query tuples, to capture a user's query intent. It then efficiently finds top-ranked approximate answer graphs and answer tuples.**

I. Introduction

There is an unprecedented proliferation of *knowledge graphs* that record millions of entities (e.g., persons, products, organizations) and their relationships. Numerous applications are tapping into such graphs in domains such as search, recommendation systems, business intelligence and health informatics. These graphs are often stored in relational databases, graph databases and triplestores. In retrieving data from these databases, the norm is to use structured query languages such as SQL, SPARQL, and those alike. However, writing structured queries requires extensive experience in query language, data model, and good understanding of particular datasets [1].

Motivated by the aforementioned usability challenge, we build GQBE (Graph Query by Example) [2], [3], a system that queries knowledge graphs by example entity tuples instead of graph queries. Given a knowledge graph and a query tuple consisting of entities, GQBE finds similar answer tuples. Consider the scenario where a Silicon Valley business analyst wants to find entrepreneurs who founded technology companies headquartered in California. Suppose she knows an example query tuple such as ⟨Jerry Yang, Yahoo!⟩ that satisfies her query intent. Given such an example tuple as input to GQBE, the answer tuples can be ⟨Steve Wozniak, Apple Inc.⟩ and ⟨Sergey Brin, Google⟩, which are founder-company pairs. If the query tuple consists of 3 or more entities (e.g., ⟨Jerry Yang, Yahoo!, Sunnyvale⟩), the answers will be similar tuples of the same cardinality (e.g., ⟨Steve Wozniak, Apple Inc., Cupertino⟩). GQBE also supports multiple query tuples as input which collectively better capture the user intent, and further details of the same can be found in [3].

Related Work: Substantial progress has been made on query mechanisms that help users construct query graphs or even do not require explicit query graphs. Paradigms such as keyword-based query formulation [4], [5], interactive and form-based query formulation [6], [7], and approximate graph query [8] require effort from users to convey the query intent. For instance, using keyword-based methods, not only a user may find it challenging to clearly articulate a query, but it is also non-trivial for a query system to precisely separate these keywords and correctly match them with entities, entity types and relationships. In contrast, a GQBE user only needs to know the names of some entities in example tuples, without being required to specify how exactly the entities are related. EQ [9] proposes the concept of exemplar queries which is similar to the paradigm of GQBE.

II. Graph Query By Example

Problem Statement: A knowledge graph G is a directed multi-graph with node set $V(G)$ and edge set $E(G)$. Each node $v \in V(G)$ represents an entity and each labeled edge $e=(v_i, v_j) \in E(G)$ denotes a directed relationship from entity v_i to entity v_j. A tuple $t=\langle v_1, \ldots, v_n \rangle$ is an ordered list of entities in G. In the aforementioned Silicon Valley analyst example, $t=\langle$Jerry Yang, Yahoo!$\rangle$. Given a knowledge graph G and a query tuple t, our goal is to find the top-k similar answer tuples t'.

Figure 1 depicts the overall architecture of GQBE. The input to GQBE is only an example query tuple. The query graph discovery module automatically discovers a *maximum query graph* (MQG) to approximately capture the user's query intent. It is unlikely to find answer graphs exactly matching the MQG, so the answer space modeling and query processing components find approximately matching answer graphs efficiently. Various algorithms and other details can be found in [3], while we only provide a brief overview here due to space limitations.

Maximum Query Graph Discovery: Edges are weighted to capture the importance of relationships, using several distance-based and frequency-based heuristics. The weight $\mathsf{w}(e)$ of an edge $e=(u, v)$ is 1) directly proportional to its inverse edge frequency, $\mathsf{ief}(e)$, which captures how rare a relationship is in the data graph, 2) inversely proportional to its participation, $\mathsf{p}(e)$, which determines the number of edges in the data graph that share the same label and one of e's end nodes (u or v), and 3) inversely proportional to $\mathsf{d}(e)$, the distance of edge e from the query entities.

A greedy heuristic is used to discover the MQG that captures important relationships while being reasonably small, since the problem of finding an m-edged graph containing all query entities while maximizing the total edge weight is NP-hard [3]. An example MQG for ⟨Jerry Yang, Yahoo!⟩ is shown in Fig. 1.

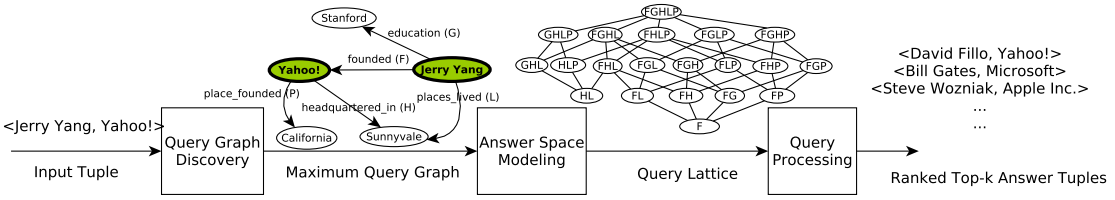


Fig. 1. The Architecture and Components of GQBE

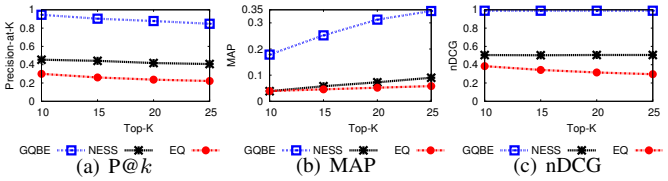
Query	PCC	Query	PCC	Query	PCC	Query	PCC
F_1	0.79	F_2	0.78	F_3	0.60	F_4	0.80
F_5	0.34	F_6	0.27	F_7	0.06	F_8	0.26
F_9	0.33	F_{10}	0.77	F_{11}	0.58	F_{12}	undefined
F_{13}	undefined	F_{14}	0.62	F_{15}	0.43	F_{16}	0.29
F_{17}	0.64	F_{18}	0.30	F_{19}	0.40	F_{20}	0.65

TABLE I. PEARSON CORRELATION COEFFICIENT (PCC) BETWEEN GQBE AND AMAZON MECHANICAL TURK WORKERS, k=30

Answer Space Modeling and Query Processing: To find approximate matches to the MQG, GQBE models the space of all answer graphs as a query lattice formed by the subsumption relationship between all subgraphs of the MQG. The query lattice for ⟨Jerry Yang, Yahoo!⟩ is shown in Fig. 1. The top-most node in the lattice is the MQG, and the bottom-most nodes are called *minimal query trees* which are trees that connect all the query entities in the MQG. An approximate answer graph is defined as an edge-isomorphic match to some query graph (answer tuples are projected from these answer graphs), which is a subgraph of the MQG, and is present in the query lattice as a lattice node. We employ an upper-bound based *bottom-up*, *best-first* strategy to explore the lattice, that allows sharing of computation. We start with evaluating the minimal query trees, as multi-way join queries. After a query graph is processed, its answers are materialized in files. To process a query Q that is not a minimal query tree, at least one of its children Q'=$Q-e$ must have been processed, whose results are used to quickly perform a right-deep hash-join. The node that has the best upper-bound is always chosen to evaluate next. Every time a lattice node returns no matching answer graphs, all of its super-graphs are pruned and the lattice changes dynamically. The algorithm terminates when the current score of the k^{th} best answer tuple so far is greater than the upper-bound score of the next best lattice node chosen by the algorithm, whose correctness is guaranteed by a theorem [3] that states that we cannot get any answer tuple better than the current top-k by executing any other unevaluated node in the lattice.

III. EXPERIMENTS

We evaluated GQBE using a preprocessed Freebase data graph containing 28M nodes, 47M edges and 5,428 distinct edge labels. We evaluated 20 queries on this graph and obtained the top-30 ranked answers from GQBE. User studies were conducted on Amazon Mechanical Turk to study the quality of this ranking, using Pearson Correlation Coefficient (PCC). A PCC value in the range of 0 to 1 indicates a positive correlation with users' preferences. PCC is undefined when all entries in a list have the same rank. Table I shows that GQBE attained a positive correlation on 18 queries. We also compared the accuracy of GQBE with NESS [8] and EQ [9]



(a) P@k (b) MAP (c) nDCG

Fig. 2. Accuracy of GQBE, NESS and EQ over 11 Freebase Queries

using three measures: precision-at-k (P@k), mean average precision (MAP), and normalized discounted cumulative gain (nDCG). NESS is a graph querying framework that finds approximate matches of query graphs with unlabeled nodes which correspond to query entity nodes in MQG. NESS does not consider edge-labeled graphs. We adapted it by requiring each candidate node v' of v to have at least one incident edge in the data graph bearing the same label of an edge incident on v in the MQG. EQ does not provide a definitive way of discovering query graph given an exemplar query tuple. Therefore, we provided the MQG discovered by GQBE as the input query graph to EQ. The query processing in EQ is similar to NESS, but it requires answer graphs to exactly match the query graph structure. Figure 2 shows that GQBE outperforms NESS and EQ on all three measures. Only 11 of the 20 queries were considered for this experiment since no answer tuples were returned by EQ for the rest of them. Experiments highlighting the improved accuracy of GQBE with multi-tuple queries, efficient query processing, accuracy results over the DBpedia dataset, among others, can be found in [3].

REFERENCES

[1] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu, "Making database systems usable," in *SIGMOD*, 2007.

[2] N. Jayaram, M. Gupta, A. Khan, C. Li, X. Yan, and R. Elmasri, "GQBE: querying knowledge graphs by example entity tuples," in *ICDE*, 2014.

[3] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri, "Querying knowledge graphs by example entity tuples," *TKDE*, vol. 27, no. 10, pp. 2797–2811, 2015.

[4] J. Pound, I. F. Ilyas, and G. E. Weddell, "Expressive and flexible access to web-extracted data: a keyword-based structured query language," in *SIGMOD*, 2010.

[5] J. Yao, B. Cui, L. Hua, and Y. Huang, "Keyword query reformulation on structured data," *ICDE*, 2012.

[6] E. Demidova, X. Zhou, and W. Nejdl, "Freeq: an interactive query interface for freebase," in *WWW*, 2012.

[7] M. Jarrar and M. D. Dikaiakos, "A query formulation language for the data web," *TKDE*, vol. 24, no. 5, pp. 783–798, 2012.

[8] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao, "Neighborhood based fast graph search in large networks," in *SIGMOD*, 2011.

[9] D. Mottin, M. Lissandrini, Y. Velegrakis, and T. Palpanas, "Exemplar queries: Give me an example of what you need," in *VLDB*, 2014.