Intuitive and Interactive Query Formulation to Improve the Usability of Query Systems for Heterogeneous Graphs

Nandish Jayaram
Supervised by Chengkai Li and Ramez Elmasri
University of Texas at Arlington
nandish.jayaram@mavs.uta.edu

ABSTRACT

Heterogeneous graphs are increasingly used to represent complex relationships in schema-less data. Querying these graphs is a fundamental and critical task for many graph applications. While such graphs are content-rich, they are difficult to use. In this work we present techniques to help schema-agnostic users easily query such large heterogeneous graphs. As an initial step towards making such graphs more usable, we propose two systems: (1) GQBE, a system which supports a new querying paradigm that queries such graphs by example entity tuples, without requiring them to form complex graph queries, and (2) VIIQ, an interactive visual query formulation system that helps users construct exact query graphs. VIIQ ranks the labels for manually added query graph components, and also automatically recommends new edges to include in the query graph, based on how likely they will be of interest to the user.

1. INTRODUCTION

There is an unprecedented proliferation of heterogeneous graph data in our society today, that are useful for numerous applications, including search, recommendation, and business intelligence. Graphs represent complex relationships in data with heterogeneous and ever-changing schema, such as Freebase, DBpedia and YAGO. Figure 1 is an excerpt of such a graph where nodes represent entities and labeled edges represent relationships between entities. Such graphs are often stored in relational databases, triplestores and graph databases. Given such a large heterogeneous graph, being able to easily query it is a fundamental problem and a critical task for many graph applications.

Both users and application developers are often overwhelmed by the daunting task of understanding and using heterogeneous graphs, due to the sheer size and complexity of such data. More specifically, the challenges lie in the gap between complex data and non-expert users. Query graphs and structured query languages such as SQL, SPARQL, and those alike are often used to specify the exact query intent for such data. However, formulating such query graphs or structured queries require extensive experiences in query language, data model, and a good understanding of particular datasets [4]. For instance, consider the scenario where a computer historian is interested in preparing an article on Turing Award winning American university professors who have also designed a programming language. Formulating even a simple query graph that captures this query intent (as shown in Fig. 2) requires a commanding knowledge of the complex schema, and is difficult for both novice and experienced users alike.

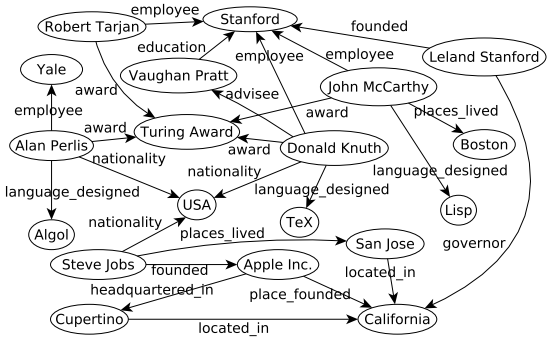


Figure 1: An Excerpt of a Heterogeneous Graph

Motivated by the aforementioned usability challenges, my PhD dissertation focuses on addressing the problem of improving the query formulation capability of query systems for large heterogeneous graphs. Figure 2 shows the architecture of the proposed framework. More specifically, we present two different techniques: 1) GQBE (**G**raph **Q**uery **B**y **E**xample), a system that supports a new querying paradigm that queries graphs by example entity tuples, instead of query graphs. GQBE lets schema-agnostic users provide example tuples as input to obtain similar answer tuples as output. The *query graph discovery* component shown in Fig. 2 automatically discovers a hidden query graph that tries to capture the query intent behind the example query tuples, and 2) VIIQ (**V**isual **I**nterface for **I**nteractive graph **Q**uery formulation), a system that helps schema-agnostic users formulate query graphs specifying their exact query intent. The *query canvas* component of VIIQ shown in Fig. 2, provides an interactive interface for users to formulate their query graph in. VIIQ helps users in the query formulation process by automatically making suggestions that are ranked by how likely they are relevant to the user's query intent [1].

GQBE [6, 7] is among the first to query heterogeneous graphs by example entity tuples. Given a data graph and one or more example query tuples consisting of entities, GQBE finds similar answer tuples. Suppose the historian knows an example query tuple such as ⟨Donald Knuth, Stanford, TeX⟩ that satisfies her query intent. The

[1] Demonstration videos of GQBE and VIIQ can be found at `http://www.youtube.com/watch?v=4QfcV-OrGmQ` and `https://youtu.be/el_w1vEvtoA` respectively.

Proceedings of the VLDB 2015 PhD Workshop

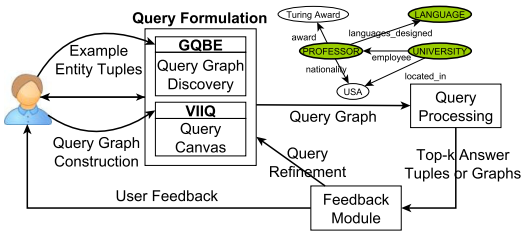


Figure 2: Framework for Querying Heterogeneous Graphs

answer tuples can be ⟨John McCarthy, Stanford, Algol⟩ and ⟨Barbara Liskov, MIT, CLU⟩, which are professor-university-language triples. The user need not specify *how* various entities in the example tuple are related. Instead, the system discovers a query graph that tries to capture relationships that may be relevant to the query intent.

VIIQ, on the other hand, helps users *easily* formulate *exact* query graphs. VIIQ provides a visual interface that enables users to easily construct query graph components. To help schema-agnostic users specify their exact query intent, VIIQ operates in *passive* and *active* modes. By default VIIQ operates in passive mode. Based on the partially constructed query graph, the system automatically suggests top-k new edges that may be relevant to the user's query intent, without being triggered by any user actions. The active mode is triggered when the user adds new nodes or edges to the partial query graph. For a newly added edge, the suggested edge labels are ranked based on the likelihood of their relevance to the user's query intent. The graph in Fig. 2 can be constructed iteratively with the help of suggestions made by VIIQ. To the best of our knowledge, VIIQ is the first visual query formulation system that makes ranked suggestions to help users construct exact query graphs.

Once a query graph is formed, the *query processing* component shown in Fig. 2 evaluates the query graph to find answers, which are answer tuples for GQBE and answer graphs for VIIQ. The *feedback module* in Fig. 2 presents these answers to the user to obtain their feedback on the relevance of the results, and further refine the query graph if necessary.

2. 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 [11, 13], natural language questions [12], interactive and form-based query formulation [2, 5], and approximate graph query [9] require effort from users to convey the query intent. For instance, using keyword-based methods, a user has to articulate query keywords, e.g., "Turing award winning American professors and languages designed by them" for the aforementioned historian. Not only a user may find it challenging to clearly articulate a query, but also a query system might not return accurate answers, since it is non-trivial to precisely separate these keywords and correctly match them with entities, entity types and relationships. This has been verified through our own experience on a keyword-based system adapted from SPARK [10]. 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. Alternatively, paradigms such as keyword-based querying are more useful when corresponding example tuples are unknown to the user.

Several graph query systems allow users to construct query graphs through a visual interface [3, 1, 8]. But, since the focus of these systems is query processing, their query formulation components are limited to only being a graphical platform to add nodes and edges with ease using mouse and keyboard actions. Little help is offered to *easily* choose the labels of various components in a query graph. With large heterogeneous graphs, every time a new query component is added, users are inundated with possibly hundreds of or more options for the new component's label, sorted alphabetically. It is a daunting task to browse through all the options to select the appropriate label to add. Existing systems help users specify queries either easily *or* exactly, but not both. In contrast, VIIQ helps users easily construct exact query graphs by ranking candidates for newly added query graph components in active mode, and automatically suggesting new relevant edges to include in the query graph in passive mode, without any input from the user.

3. GRAPH QUERY BY EXAMPLE (GQBE)

GQBE lets users query large graphs by example entity tuples, and Fig. 3 shows the user interface of GQBE. The search bar, assisted by user interface tools such as auto-completion in identifying the exact entities in the data graph, is used to enter the entities of an example tuple. To better communicate the query intent, the '+' button can be used to provide multiple query tuples. Since the user is not required to specify how these entities are related, the query graph discovery component of GQBE automatically derives a *maximum query graph* (MQG) to approximately capture the user's query intent, which can be viewed by clicking the *view maximum query graph* button in Fig. 3. There can be a large space of approximate answer graphs since it is unlikely to find answer graphs exactly matching the MQG. The query processing component models the space of answer graphs by a query lattice formed by the subsumption relation between all possible subgraphs of the MQG. A top-k lattice exploration algorithm that only partially evaluates the lattice nodes in the order of their corresponding query graphs' upper-bound scores is employed to evaluate the lattice. The ranked answer tuples obtained are displayed as shown in Fig. 3, and their corresponding matching answer graphs can be viewed by clicking on the *view answer graph* button. Various algorithms and other details of the query processor can be found in [7], while we only provide a brief overview here.

Maximum Query Graph Discovery: Edges are weighted to capture 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, $\mathsf{ief}(e)$, that captures how rare a relationship is in the data graph, 2) inversely proportional to its participation, $\mathsf{p}(e)$, that 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 the distance, $\mathsf{d}(e)$, that captures the distance of edge e from the query entities.

The MQG must be reasonably small while capturing important relationships. A greedy heuristic is used to capture the MQG, since the problem of finding such an m-edged graph containing all query entities while maximizing the total edge weight is NP-hard [7].

Query Processing: In order to find approximate matches to the maximum query graph, GQBE models the space of all answer graphs as a query lattice formed by the subsumption relationship between all subgraphs of the MQG. 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. We start evaluating from the minimal query trees, and always choose to evaluate the node that has the best upper-bound, which is essentially the

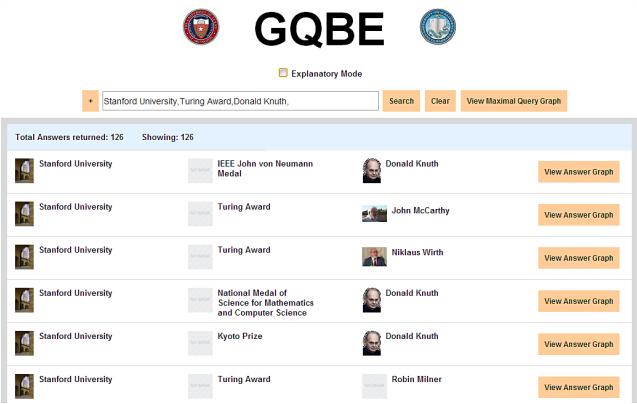


Figure 3: GQBE**'s User Interface**

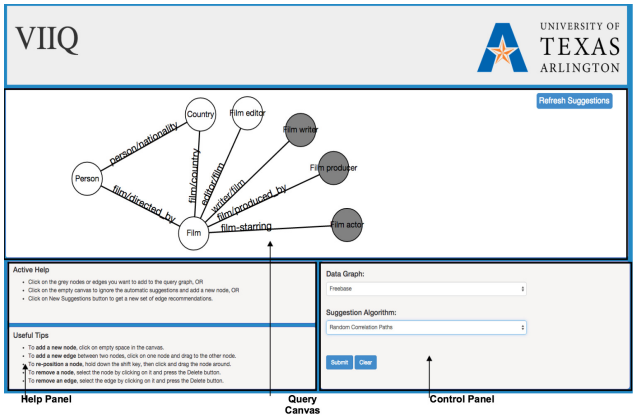
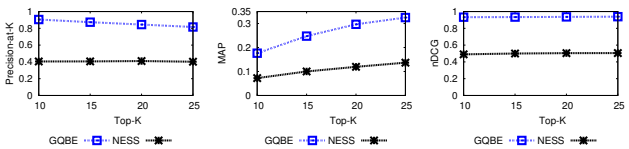


Figure 4: VIIQ**'s User Interface**

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 1: Pearson Correlation Coefficient (PCC) between GQBE **and Amazon Mechanical Turk Workers,** k=30



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

Figure 5: Accuracy of GQBE **and** NESS **on Freebase Queries**

largest super-graph in the lattice. 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 [7] that states that we cannot get any answer tuple better than the current top-k by executing any other unevaluated node in the lattice.

Experiments: We evaluated GQBE (and VIIQ) 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 with Amazon Mechanical Turk to study the quality of this ranking, using Pearson Correlation Coefficient (PCC). A PCC value in the range 0 to 1 indicates a positive correlation with users' preferences. PCC is undefined when all entries in a list have the same rank. Table 1 shows that GQBE attained a positive correlation on 18 queries. We also compared the accuracy of GQBE with NESS [9] 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. Figure 5 shows that GQBE outperforms NESS on all three measures.

4. INTERACTIVE QUERY GRAPH COMPLETION (VIIQ)

VIIQ helps users easily formulate query graphs specifying their exact query intent. Figure 4 shows the user interface of VIIQ, where users can formulate query graphs on the *query canvas*, while various parameters of the system are tuned in the *control panel*. As mentioned earlier, VIIQ supports two modes of operation, passive and active. VIIQ operates in passive mode by default. Top-k edges that may be relevant to the user's query intent are automatically suggested, without being triggered by any user actions. When the user adds new nodes or edges on the canvas by simple mouse actions, VIIQ operates in active mode. A set of candidate labels C is determined for query graph components during both modes of operation. These candidate labels are ranked based on their relevance to the user's query intent and presented to the user.

In passive mode, based on the connected partial query graph on the canvas, the system automatically suggests top-k new edges relevant to the user. The new edges suggested are incident on the partial query graph in the canvas, and connected to grey nodes as shown in Fig. 4. The user can click on some grey nodes to add them to the query graph, and ignore the others. The unselected grey nodes are deleted with a mouse click on the canvas, and the next set of new suggestions are automatically displayed. If none of the suggestions obtained in passive mode are useful and the user does not select any grey nodes, a new set of suggestions can be manually triggered using the *refresh suggestions* button on the query canvas.

Users can click on the canvas to add a new node, and VIIQ switches to active mode. A suggestion panel pops up when a new node is added. Nodes in a heterogeneous graph represent entities. Real world entities, and thus their labels, can be grouped into a natural hierarchy of domains, types and entities, where multiple entities may belong to the same type and multiple types may belong to a single domain. We use such ontological hierarchy to help users navigate through the options for a node label. Users can either select a type, or entity value as the node label using drop-down lists, where options are sorted alphabetically. A new edge can be added in active mode by clicking on one node and dragging the mouse to the destination node. The possible labels for the newly added edge are ranked by their relevance to the query intent and displayed using a drop-down list in a pop-up suggestion panel.

The assistance provided by VIIQ during query formulation mainly consists of edge suggestions made to the user. Given a set of candidate edges C, we must rank these edges based on the likelihood of them being accepted by the user, since ranking relevant edges

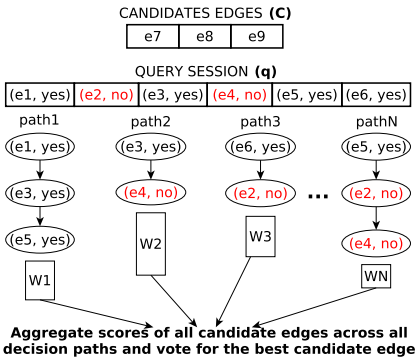


Figure 6: Ranking Based on Random Correlation Paths

higher is considered important. The likelihood of a candidate edge being accepted is conditioned on the various edges suggested and their corresponding user responses obtained hitherto, which are captured in query session q. Edges accepted and added to the partial query graph are called *positive* edges. In passive mode, the suggested edges not relevant to the user, called *negative* edges, are ignored by clicking on the canvas. A query log W that captures many such query sessions is useful in gauging the user's query intent and ranking candidate edges for a new query session q. We simulate such a query log using Wikipedia and the data graph.

Ranking Based on Random Correlation Paths: Edges in C must be ranked based on the correlation strength between an edge $e \in C$ and q. One way to measure this correlation strength is using the support we find for q in query log W, which are the query sessions in W that subsume query session q. One can assume strict correlation between all edges in q, but for a long q, this may lead to zero support in W. The other extreme is to assume independence between all edges in q (like in a naive Bayes classifier), but this will likely lead to a large noisy support in W. We propose to find *random correlation paths* that capture the correlation between only a subset of edges in q, striking a balance between the aforementioned extremes of considering correlation between edges in q.

A correlation path $\overrightarrow{q}$ for a given query session q, is the ordered set of edges in q. We define $supp(\overrightarrow{q})$, the support for a correlation path $\overrightarrow{q}$, as the number of entries in W that are supersets of q. A postfix of $\overrightarrow{q}$, denoted $postfix(\overrightarrow{q}, e_{k+1})$, is the new path formed by adding edge e_{k+1} to $\overrightarrow{q}$. If $\overrightarrow{q}$=$\{e_1, e_2, \ldots, e_{k-1}, e_k\}$, then $postfix(\overrightarrow{q}, e_{k+1})$=$\{e_1, e_2, \ldots, e_{k-1}, e_k, e_{k+1}\}$. In order to rank the candidate edges, we build $\Re$, a set of N random correlation paths as shown in Fig. 6. The query session in Fig. 6 has edges e_1-e_6 and the candidate edges are e_7-e_9. The edges with a yes denote positive edges, and edges with a no denote negative edges in q. $\Re$ consists of the shortest correlation paths based only on those edges in q whose supports are no more than a threshold τ. All edges $e \in C$ are ranked by the final score $score(e)$, given by

$$score(e) = \frac{1}{|\Re|} \times \sum_{\overrightarrow{p} \in \Re} \frac{supp(postfix(\overrightarrow{p}, e))}{supp(\overrightarrow{p})}$$

Experiments: Preliminary experiment results, over Freebase data graph, suggest that ranking candidates by this approach is significantly better than both the methods (one based on strict correlation, and the other on naive Bayes classifier). 9 target query graphs, each with up to 5 edges were designed. The system operated only in passive mode and the top-1 edge was suggested in each iteration. The number of iterations required to reach the target graph starting from a single-edge partial query graph was measured. 7 out of the 9 target query graphs were achieved within 21 suggestions (on average) with our proposed method, while not a single relevant edge was suggested by the other two methods for 8 of these 9 query graphs.

5. CONCLUSION AND FUTURE WORK

Querying large heterogeneous graphs with ever-changing schema is a difficult task, since existing graph query systems and paradigms are either difficult to use or cannot be used to formulate exact queries. My PhD dissertation addresses the problem of improving the usability of query systems for such graphs. We propose two systems: 1) GQBE, that supports a new querying paradigm that queries such graphs by example entity tuples, without the user having to form query graphs, and 2) VIIQ, a visual interface that helps users formulate query graphs by automatically suggesting relevant edges to add in passive mode, or by ranking labels for explicitly added query components in active mode.

The future direction to complete my PhD dissertation is to focus on the feedback module of the framework shown in Fig. 2. Users can mark the relevance of the top-k answer tuples obtained from GQBE, which can then be used to refine the MQG and obtain better results. For VIIQ, the feedback on the relevance of answer graphs can be used to refine the query graph and find better answers. Another extension is to evaluate partial query graphs and guide users towards formulating query graphs that are likely to produce results.

Acknowledgments The author would like to thank Mahesh Gupta and Sidharth Goyal for their contribution in implementing the systems' GUI. The author has been partially supported by NSF grants IIS-1018865, CCF-1117369 and IIS-1408928. Any opinions, findings, and conclusions in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

6. REFERENCES

[1] D. H. Chau, C. Faloutsos, H. Tong, J. I. Hong, B. Gallagher, and T. Eliassi-Rad. GRAPHITE: A visual query system for large graphs. In *ICDM*, 2008.

[2] E. Demidova, X. Zhou, and W. Nejdl. FreeQ: an interactive query interface for Freebase. In *WWW*, demo paper, 2012.

[3] H. H. Hung, S. S Bhowmick, B. Q. Truong, B. Choi, and S. Zhou. Quble: Blending visual subgraph query formulation with query processing on large networks. SIGMOD, 2013.

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

[5] M. Jarrar and M. D. Dikaiakos. A query formulation language for the data web. *TKDE*, 24:783–798, 2012.

[6] N. Jayaram, M. Gupta, A. Khan, C. Li, X. Yan, and R. Elmasri. GQBE: Querying knowledge graphs by example entity tuples. In *ICDE (demo description)*, 2014.

[7] N. Jayaram, A. Khan, C. Li, X. Yan, and R. Elmasri. Querying knowledge graphs by example entity tuples. *IEEE Transactions on Knowledge and Data Engineering*, (to appear).

[8] C. Jin, S. S. Bhowmick, B. Choi, and S. Zhou. prague: A practical framework for blending visual subgraph query formulation and query processing. In *ICDE*, 2012.

[9] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *SIGMOD'11*.

[10] Y. Luo, X. Lin, W. Wang, and X. Zhou. Spark: top-k keyword query in relational databases. In *SIGMOD*, 2007.

[11] 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*, pages 423–434, 2010.

[12] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Deep answers for naturally asked questions on the web of data. In *WWW*, demo paper, pages 445–449, 2012.

[13] J. Yao, B. Cui, L. Hua, and Y. Huang. Keyword query reformulation on structured data. *ICDE*, pages 953–964, 2012.