VIIQ: Auto-Suggestion Enabled Visual Interface for Interactive Graph Query Formulation

Nandish Jayaram, Sidharth Goyal, Chengkai Li

University of Texas at Arlington

ABSTRACT

We present VIIQ (pronounced as wick), an interactive and iterative visual query formulation interface that helps users construct query graphs specifying their exact query intent. Heterogeneous graphs are increasingly used to represent complex relationships in schemaless data, which are usually queried using query graphs. Existing graph query systems offer little help to users in easily choosing the exact labels of the edges and vertices in the query graph. VIIQ helps users *easily* specify their *exact* query intent by providing a visual interface that lets them graphically add various query graph components, backed by an edge suggestion mechanism that suggests edges relevant to the user's query intent. In this demo we present: 1) a detailed description of the various features and user-friendly graphical interface of VIIQ, 2) a brief description of the edge suggestion algorithm, and 3) a demonstration scenario that we intend to show the audience.

1. INTRODUCTION

There is an unprecedented proliferation of heterogenous graph data in our society today. These graphs are increasingly used to represent complex relationships in schema-less data such as Freebase, DBpedia and YAGO. Fig.1 is an excerpt of such a graph where nodes represent entities and labeled edges represent relationships between entities. Given such a large heterogenous graph, being able to easily query it is a fundamental problem and a critical task for many graph applications. Query graphs are often used to specify the query intent for such graphs. But, formulating these query graphs is a daunting task since it requires users to know a vocabulary comprised of many labels and types of nodes and edges.

Several graph query systems allow users to construct query graphs through a visual interface [4, 3, 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. There are other querying paradigms [1, 9, 6, 7, 10] that help users query graph data. Declarative languages like SPARQL [1] are used to exactly specify query intent, but present a usability barrier [5]. Paradigms such as keyword search, approximate graph query [9] and query-by-example [6, 7, 10] can simplify query formulation, but cannot be used to specify users' exact query intent. In summary, existing systems help users specify queries either easily *or* exactly, but not both.

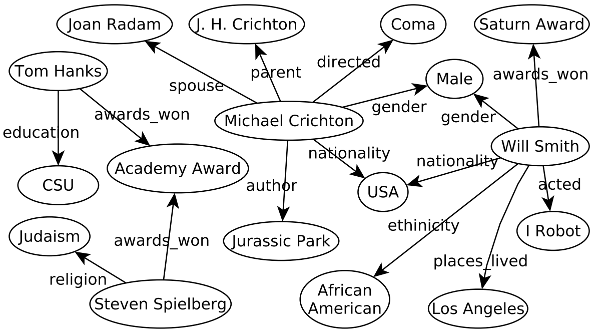
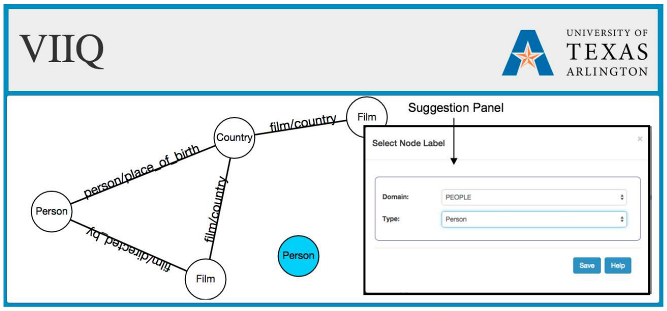


Figure 1: Excerpt of a heterogeneous graph

To this end, we propose VIIQ (**V**isual **I**nterface for **I**nteractive graph **Q**uery formulation), a system that helps users *easily* formulate *exact* query graphs. VIIQ provides a visual interface that enables users to easily construct various query graph components. To allow schema-agnostic users to specify their exact query intent, the candidate label suggestions for a newly added query graph component are ranked on how likely they will be of interest to the user. To further minimize the burden on users, VIIQ automatically suggests new edges to include into the query graph without the user manually adding new query components. A visual querying interface that intelligently helps users formulate query graphs is acknowledged as an important step towards superior consumption and management of graph data [2]. To the best of our knowledge, VIIQ is the first visual query formulation system that actively makes ranked suggestions to help users construct exact query graphs.

VIIQ supports two modes of operation, *active* and *passive*. When the user adds new nodes or edges onto a canvas by simple mouse actions, VIIQ operates in active mode. For a newly added node, the suggested labels are displayed hierarchically in a pop-up box, as shown in Fig. 2, where type Person is chosen as the label for the node. For a newly added edge, the suggested edge labels are ranked based on the likelihood of their relevance to the user's query intent. Fig. 3 shows the ranked suggestions for the newly added edge between nodes Person and Film. When the user is not operating on anything, VIIQ switches into passive mode. The system automatically recommends top-k new edges that may be relevant to



Domain: PEOPLE

Type: Person

Figure 2: Adding Node in Active Mode

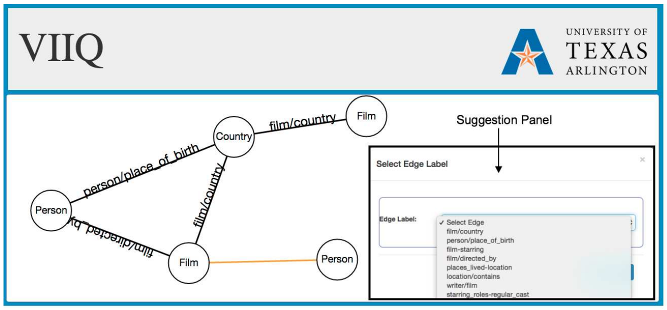


Figure 3: Adding Edge in Active Mode

the user's query intent, without being triggered by any user actions. Fig. 4 shows the snapshot of a partially constructed query graph, with nodes and edges suggested in passive mode. The nodes in grey and the edges incident on them are the new automatic suggestions made by the system.

2. USER INTERFACE

Figure 4 shows the graphical user interface of VIIQ. The system provides several functionalities that aid users in constructing query graphs: 1) a canvas for formulating the query graph, which includes drawing query graph components or selecting automatically made suggestions, 2) an active mode of operation where users can add new nodes and edges using simple mouse actions, and 3) a passive mode of operation where the system automatically suggests new edges to add based on their relevance to the user's query intent.

There are mainly four GUI components in VIIQ. **Query Canvas** is the area used to construct the query graph. New nodes and edges are added here in active mode using simple mouse actions. New top-k edges are also automatically suggested and displayed on the canvas in passive mode. The **Suggestion Panel**, as shown in Figs. 2 and 3, is a pop-up box that displays label suggestions for newly added nodes and edges in active mode. The suggested labels are ranked and displayed using drop-down lists. The **Control Panel** is used to tune various parameters of the system. The drop-down list under Data Graph is used to select the underlying data graph one wishes to query. The drop-down list under Suggestion Algorithm is used to specify the edge suggestion algorithm to use. Finally, the **Help Panel** displays general tips to operate the system. It also dynamically displays messages explaining the allowable user actions at any given moment in the query formulation process.

As mentioned earlier, VIIQ operates in active and passive modes. A user can click on any empty part of the canvas to add a new node. A suggestion panel pops up when a new node is added as shown in Fig. 2. 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 an exact entity value as the node label using drop-down lists in the Suggestion Panel. 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 the suggestion panel as shown in Fig. 3.

Passive mode of operation kicks in whenever a user is idle and there is a connected partial query graph on the canvas. In passive mode, 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 Fig. 4 shows an example instance where the top-3 new edges (incident on nodes shaded grey) are the automatic suggestions made. 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.

The new edge suggestions made are based on the partial query graph formed hitherto. The ranking of suggested edge labels in both active and passive mode depends on the underlying edge suggestion algorithm which is briefly described next.

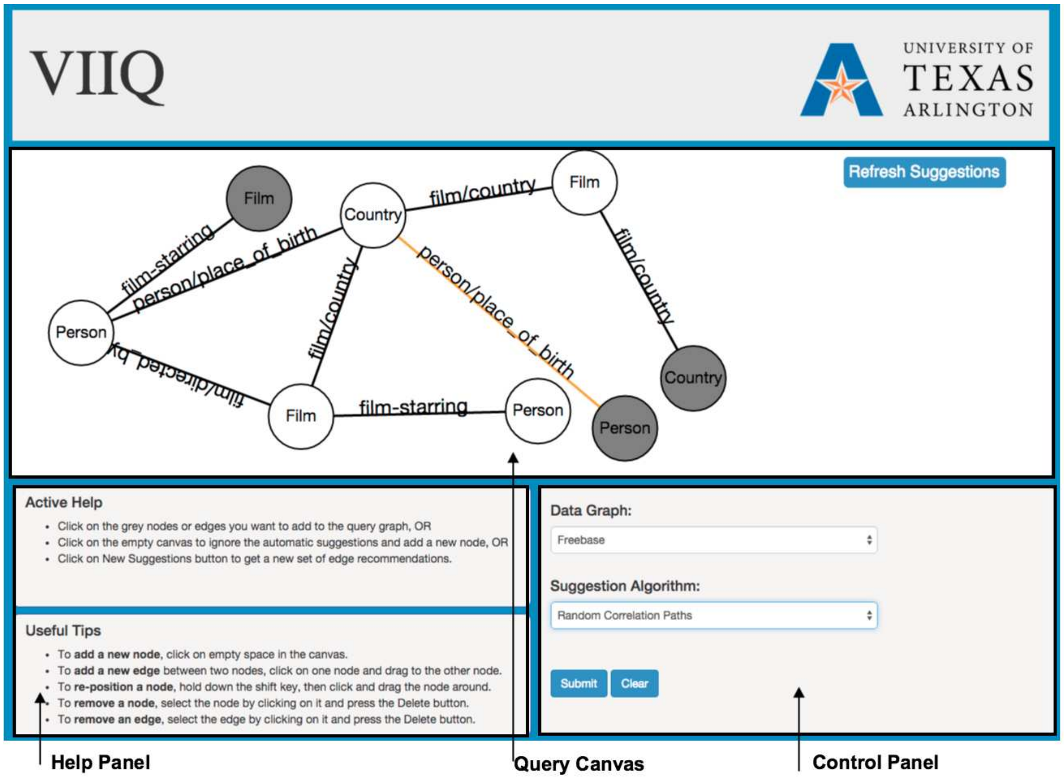
3. RANKING CANDIDATE EDGES

This section provides a brief overview of VIIQ's underlying edge suggestion algorithm.

A data graph G_d is a connected, directed, labeled multi-graph with node set $V(G_d)$ and edge set $E(G_d)$. Each node $v \in V(G_d)$ is labelled by its unique ID and belongs to one or more entity types (e.g., Person and Actor). All entity types form a set T_V. Each edge $e \in E(G_d)$ is labelled by its type (e.g., *directed*). The target query graph that represents the user's intent is a connected graph Q_t. The nodes in Q_t are either entities in $V(G_d)$ or entity types in T_V. The relationships between nodes in Q_t are defined by edge labels, i.e., edge types. For instance, an edge labeled *directed* is always from a node of type Director to a node of type Film. During the query formulation process, any connected query graph in the intermediate steps is called a partial query graph Q_p.

The assistance provided by VIIQ during query formulation mainly consists of edge suggestions made to the user. In active mode, the two ends of a newly added edge are selected by the user, and all possible edge labels between the two nodes form the set of candidate edges C. In passive mode, any edge that can potentially be incident on any node in the partial query graph Q_p is a candidate edge. The edge can be either between two current nodes in Q_p or betwen a node in Q_p and a suggested new node. Candidate edges are ranked and displayed in a drop-down list in active mode, while only the top-k edges are displayed on the canvas in passive mode.

Edges found relevant by the user, called *positive* edges, are accepted and added to the partial query graph. In passive mode, the suggested edges not relevant to the user, called *negative* edges, are ignored by clicking on the canvas. Both accepted and ignored edges play a major role in gauging the user's query intent. The query formulation process is a query session q which is a series of such suggested edges and the corresponding user responses obtained. Note that the query session q not only contains the edges forming the partial query graph, but also the edges that were rejected by the user. Given a set of candidate edges C, we must rank these edges based



Active Help

- Click on the grey nodes or edges you want to add to the query graph, OR
- Click on the empty canvas to ignore the automatic suggestions and add a new node, OR
- Click on New Suggestions button to get a new set of edge recommendations.

Useful Tips

- To **add a new node**, click on empty space in the canvas.
- To **add a new edge** between two nodes, click on one node and drag to the other node.
- To **re-position a node**, hold down the shift key, then click and drag the node around.
- To **remove a node**, select the node by clicking on it and press the Delete button.
- To **remove an edge**, select the edge by clicking on it and press the Delete button.

Data Graph: Freebase

Suggestion Algorithm: Random Correlation Paths

Figure 4: User Interface of VIIQ

on the likelihood of them being accepted by the user, since ranking relevant edges 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 is captured by the query session q.

A query log W that captures many such query sessions is useful in ranking candidate edges for a new query session q. But, such a large graph query log is not available publicly. We thus simulate a query log using Wikipedia and the data graph G_d (e.g., Freebase). For every Wikipedia page, entities occurring in each sentence are identified simply by recognizing hyperlinks to other Wikipedia pages (i.e., entities). Most nodes in data graphs like Freebase have properties such as *topic_equivalent_webpage*, that identify the Wikipedia URL corresponding to them. Such properties are used to map entities found in a Wikipeda page's sentence to nodes in G_d. The properties that connect these nodes in G_d mimic the set of positive edges in a query session. We also use data graph based statistics, by considering all properties incident on a node in G_d as such positive edges of a query session. Negative edges, which indicate edges that were ignored by the user, are injected into these simulated query sessions. If there is evidence of positive edges e_1 and e_2 in query session q_i, and another query session q_j contains e_1 but not e_2, then e_2 is injected into q_j as a negative edge. Finally, the Apriori algorithm is used to find frequent itemsets of correlated edges (query sessions) to be included in the query log W.

Problem Statement: Given a query log W, user session q so far and a set of candidate edges C, the problem is to rank edges in C by some scoring function $score(e)$.

Ranking Based on Random Correlation Paths: As mentioned earlier, the query log captures the correlation between edges. 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 $\overrightarrow{o}$ for a given set of edges o, is the ordered set of edges in o. We define $supp(\overrightarrow{o})$, the support for a correlation path $\overrightarrow{o}$, as the number of entries in W that are supersets of o. We build a random set of correlation paths consisting of only those correlation paths that are based on the current user session q. We do not attempt to pre-learn a set of correlation paths using query log W which are used to answer every arbitrary input instance (like learning a decision tree). Instead, we only build random correlation paths specific to q. This is similar to assuming a virtual space of an exponential number of decision trees built for a random forest with query log W, but instantiating only a small set of paths in these decision trees that are specific to q.

A correlation path $\overrightarrow{o}$ has a prefix path and may be associated with several postfix paths. The prefix of $\overrightarrow{o}$, denoted $prefix(\overrightarrow{o})$, is the path before adding the last edge in $\overrightarrow{o}$. A postfix of $\overrightarrow{o}$, denoted $postfix(\overrightarrow{o}, e_{k+1})$, is the new path formed by adding edge e_{k+1} to $\overrightarrow{o}$. If $\overrightarrow{o}$=$\{e_1, e_2, \ldots, e_{k-1}, e_k\}$, then $prefix(\overrightarrow{o})$=$\{e_1, e_2, \ldots, e_{k-1}\}$, and $postfix(\overrightarrow{o}, e_{k+1})$=$\{e_1, e_2, \ldots, e_{k-1}, e_k, e_{k+1}\}$.

Given a query session q and candidate edges C, each edge $e \in C$ is ranked by the support of its corresponding $postfix(\overrightarrow{q}, e)$. In order to rank the candidate edges, we build $\Re$, a set of N random correlation paths as shown in Fig. 5. The user session in Fig. 5 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

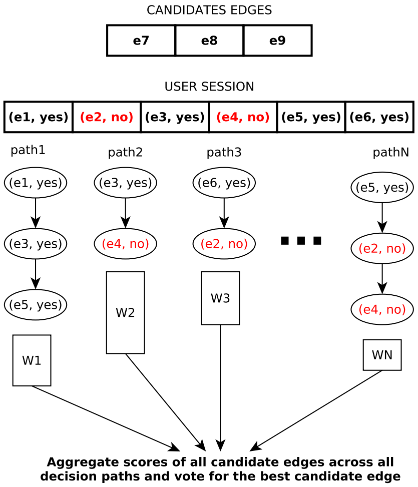


Figure 5: Ranking Based on Random Correlation Paths

edges in q. All correlation paths in $\Re$ are based only on those edges in q whose supports are no more than a threshold τ. A correlation path $\overrightarrow{p}$ is grown until $supp(\overrightarrow{p}) \leq \tau$ and $supp(prefix(\overrightarrow{p})) > \tau$, or until all the edges in q are exhausted, whichever comes first. The score of an edge $e \in C$, with regard to correlation path $\overrightarrow{p}$ is given by $score(e, \overrightarrow{p})$. All edges $e \in C$ are ranked by the final score $score(e)$, given by

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

Preliminary experiment results 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.

4. DEMONSTRATION PLAN

A demonstration video of VIIQ can be found at `https://youtu.be/el_w1vEvtoA`. In describing the demonstration scenarios, we shall assume Freebase as the data graph. In the eventual demo users will be able to choose among multiple data graphs. We use a preprocessed and cleaned Freebase data graph that contains 28M nodes, 47M edges and 5,428 distinct edge labels. The types of an entity were found using property */type/object/type*, and the domain associated with a type was obtained using the canonical name of the type. Freebase uses intermediate nodes to capture ternary and higher-arity relationships. Such relationships are replaced by multiple binary relationships (through merging edges associated with intermediate nodes), trading expressiveness for simplicity of user interface. For instance, there is an intermediate node between entities Tom Hanks and CSU (California State University) connecting properties *education* and *school*. This was replaced with a single edge labeled *education-school* as part of data pre-processing.

Scenario A: The user wishes to query Freebase and use random correlation path based edge suggestion algorithm.
(A1) Click on "Data Graph" drop-down list and select Freebase.
(A2) Click on "Suggestion Algorithm" drop-down list and select Random Correlation Paths.

Scenario B: Add new nodes in active mode.
(B1) Click on any empty space in the canvas to create a new node.
(B2) A node label suggestion panel pops up. Click on the "Domain" drop-down list and select PEOPLE.
(B3) Click on the "Type" drop-down list and select type Person.
(B4) Click on the "Save" button to apply the selected node type.
(B4) Follow steps (B1)-(B4) and add another node with domain FILM and type Film.

Scenario C: Add a new edge between two nodes in active mode.
(C1) Click on node Person and drag the mouse to node Film, or drag the mouse from Film to Person.
(C2) An edge label suggestion panel pops up, click on the "Edge Label" drop-down list and select *film/directed_by*.
(C3) Click on the "Save" button to apply the selected edge label.

Scenario D: Add an edge suggested automatically in passive mode to the partial query graph.
(D1) After performing Scenario A-Scenario C, edges and nodes suggested automatically in passive mode are displayed in grey.
(D2) Click on a newly suggested node Film Writer to add it to the partial query graph.
(D3) Click on any empty space in the canvas to save the selected node and reject the unselected grey nodes.

Scenario E: Instead of choosing the automatically suggested edges in passive mode, add a new node and edge in active mode.
(E1) After performing Scenario D, click on any empty space in the canvas to add a new nodes.
(E2) Follow steps (B1)-(B4) to add a new node with domain LOCATION and type Country.
(E3) Follow (C1)-(C3) to add a new edge labeled *person/nationality* between nodes Country and Person.

Scenario F: If none of the edges and nodes automatically suggested in passive mode are relevant, request a new set of suggestions.
(F1) After performing Scenario E, click on "Refresh Suggestions" button on the canvas and get a new set of suggestions.

5. REFERENCES

[1] SPARQL query language for RDF. `http://www.w3.org/TR/rdf-sparql-query`.

[2] S. S. Bhowmick. DB ⋈ HCI: towards bridging the chasm between graph data management and HCI. In *DEXA*, 2014.

[3] 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.

[4] 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.

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

[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. *CoRR*, abs/1311.2100, 2013.

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