Visual Graph Query Construction and Refinement

Robert Pienta, Fred Hohman, Acar Tamersoy, Alex Endert, Shamkant Navathe,
Hanghang Tong[1], Duen Horng Chau
Georgia Institute of Technology, Arizona State University[1]
{pientars, fredhohman, tamersoy, endert}@gatech.edu, sham@cc.gatech.edu,
hanghang.tong@asu.edu[1], polo@gatech.edu

ABSTRACT

Locating and extracting subgraphs from large network datasets is a challenge in many domains, one that often requires learning new querying languages. We will present the first demonstration of VISAGE, an interactive visual graph querying approach that empowers analysts to construct expressive queries, without writing complex code (see our video: https://youtu.be/l2L7Y5mCh1s). VISAGE guides the construction of graph queries using a data-driven approach, enabling analysts to specify queries with varying levels of specificity, by sampling matches to a query during the analyst's interaction. We will demonstrate and invite the audience to try VISAGE on a popular film-actor-director graph from Rotten Tomatoes.

CCS Concepts

•**Human-centered computing** → **Visualization systems and tools;** •**Information systems** → *Search interfaces;*

Keywords

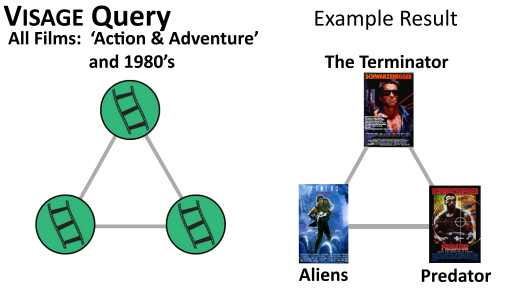
Graph querying, interactive querying, query construction

1. INTRODUCTION

From network security to bioinformatics, networks (or graphs) are often used for modeling the complex relationships among entities (e.g., who-buys-what, who-retweets-whom, etc.). Finding interesting, suspicious, or malicious patterns in these networks has been the core enabling technology for solving many important problems, such as detecting suspicious trading behavior [11], or discovering fraudsters and their accomplices in online auction sites [7]. Graph querying, subgraph matching and the emergence of new graph-database approaches have made considerable progress [4, 12, 9, 6].

Unfortunately, constructing and refining graph queries often requires learning new languages and debugging syntax errors. Users often need to overcome steep learning curves to learn querying languages specific to the graph databases storing the graphs, writing many lines of code even for conceptually simple queries, as demonstrated in Figure 1. We created VISAGE [8], the **Vis**ual **A**daptive **G**raph **E**ngine, which provides an adaptive, visual approach to graph query construction and refinement, to simplify and speed up graph query construction. VISAGE performs exact graph querying on large graphs and supports a wide variety of different node types and attributes. Our main contributions are:

- We introduce *graph-autocomplete*, an interaction technique for graphs that helps analysts construct and refine queries, and prevent over-specification from adding too many structural or feature constraints that results in too few or even no results [1]. Graph-autocomplete samples the current query in the background to stop analysts from constructing null-result-queries by analyzing and eliminating impossible extensions of the query.



Cypher Query

```
MATCH (film1:film)--(film2:film)--(film3:film),
  (film1)--(film3)
WHERE film1.genre = 'Action & Adventure' AND
  film2.genre = 'Action & Adventure' AND
  film3.genre = 'Action & Adventure' AND
  film1.year >= 1980 AND film1.year < 1990 AND
  film2.year >= 1980 AND film2.year < 1990 AND
  film3.year >= 1980 AND film3.year < 1990
RETURN film1, film2, film3
```

Figure 1: *Top*: an example Visage query for three similar 80's action films and a result. *Bottom*: the same query as above, written in the Cypher querying language. Visage's interactive graph querying approach allows analysts gradually form queries by dragging and dropping query components. It provides an alternative to writing complex graph querying code.

SIGMOD'17, May 14-19, 2017, Chicago, IL, USA

DOI: http://dx.doi.org/10.1145/3035918.3056418

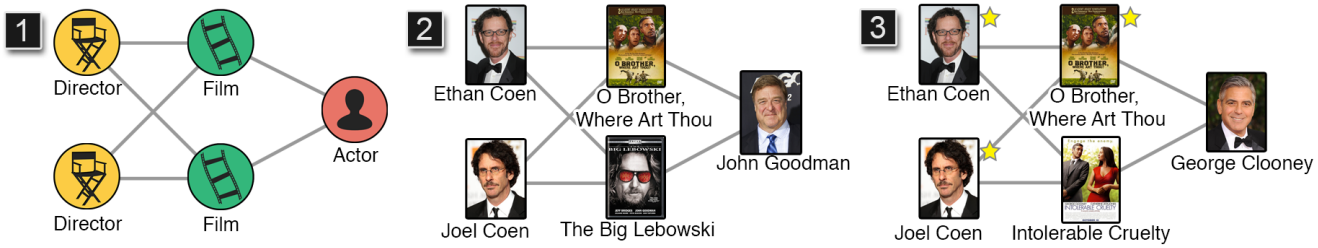


Figure 2: Visage supports many query refinement approaches for constructing a query. (1) A broad query with only node types and structure. (2) The first resulting match. (3) The Coen Brothers and the film *O Brother, Where Art Thou?* are starred, fixing these nodes; with the nodes starred, only matches with those nodes are displayed.

- We designed and developed the VISAGE system that utilizes recent advances in graph-databases to support a spectrum of querying styles, from *abstract* to *example-driven* approaches, while most other visual graph querying systems do not [5, 2, 4]. In the abstract case, analysts start with a very abstract query and narrow down the possible results by providing feature and topological constraints. In the example-driven case, originally proposed in the language *query by example* (QBE) [13], analysts can specify an exact pattern and abstract from that pattern into a query of their choice. This technique allows analysts to start from an example or keep certain values fixed in their query. In VISAGE, the analyst can *star* a node to fix its place in the query and across all of the results. We provide examples of both query-construction approaches in the Scenario Section.

We will demonstrate how graph-autocomplete can be used during the exploration process and how VISAGE's simple interface can create complex queries with minimal training.

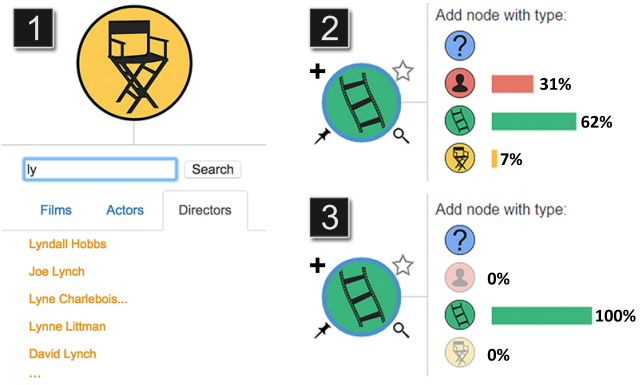


Figure 3: Each node has multiple interaction options: (1) text search for specific node values. (2) Node controls: the pin fixes the node's position; the star fixes a particular value to the node (either by text search or from selecting a particular result-node); the magnifying glass opens the text search (at 1). (3) The + adds a node and displays background-sampled, neighborhood-type distributions. Neighboring types are shown by percentage; types that do not occur in the underlying network are visibly grayed out.

2. DEMONSTRATING VISAGE

We provide a demonstration scenario both to illustrate how analysts may use VISAGE and to describe what we will show the audience. Our scenario begins with a general query of a known structure and narrows the search through query refinement.

The Rotten Tomatoes Movie Graph.

We employ a Rotten Tomatoes film-actor-director graph[1]. The graph has 58,763 nodes: 17,072 films, 8,576 directors, and 33,115 actors. There are over 468,592 undirected edges of three types: (1) film to film edges, based on Rotten Tomatoes' crowd-sourced similarity; (2) film to actor edges, showing who starred in what; (3) film to director, showing who directed what.

Demonstration Scenario.

Our analyst Bernadette wants to find co-directors who have starred the same actor in two films. She begins specifying her query in very general terms from a blank canvas. She right-clicks the background, opening a tray with each type of node. She picks a director node, she repeats this to add another director, and again to add two films and an actor. At this point her query contains just nodes, she must still add the edges connecting them. She attaches the director to the films and the films to the actor (see Figure 2-1), by clicking and dragging from one node to the other (one pair at a time). Alternatively when adding nodes, she may choose to add a neighbor to any existing node in the graph through our background sampled neighborhood menu (Figures 3-2 and 3-3). This adds a new node and edge connecting to the starting node.

She decides to add a feature condition to one of the nodes. She wants at least one of the two films to have received good reviews from the critics. She right clicks the top film in her query, opening the feature selection dendrogram (shown in Figure 5). She navigates the data-generated feature summary tree and selects "Well-rated" from the choices under the "Critics' Score" feature.

She clicks the search button. The results appear in the results list, we here show only the first result (in Figure 4-2) to save space. She likes the first result (in Figure 2-2) with the Coen Brothers, *The Big Lebowski, O' Brother Where Art Thou?*, and John Goodman. Realizing that she enjoys the work of the Coen brothers, she stars both director nodes and *O' Brother Where Art Thou?*, making them fixed values in

[1]A movie review website. http://www.rottentomatoes.com/

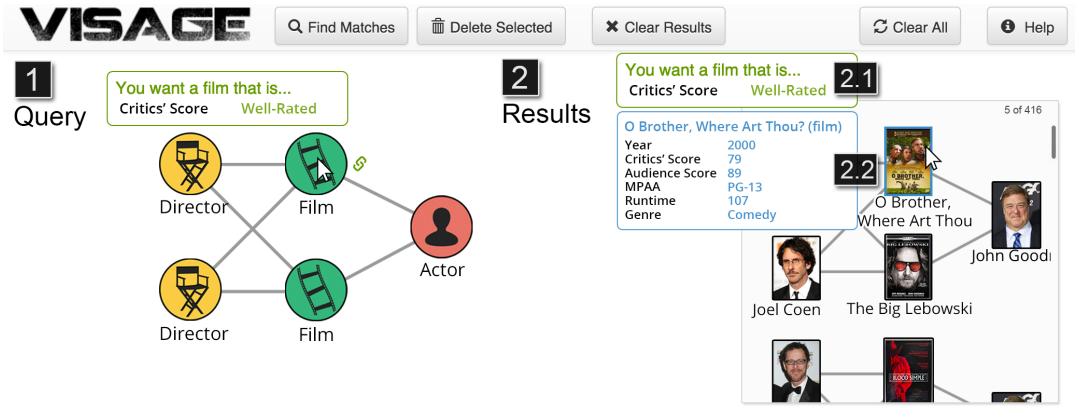


Figure 4: The interface for our demonstration shows a basic query for a film with at least one actor and one director; results are shown on the right in real time. Queries are constructed in the open space (1) by placing nodes and edges. The query results are shown in a list in (2). When a node-result is clicked, a summary of feature conditions (2.1) is shown with a summary of that node's attributes (2.2). In this example the film must have a critics' score of "Well-rated".

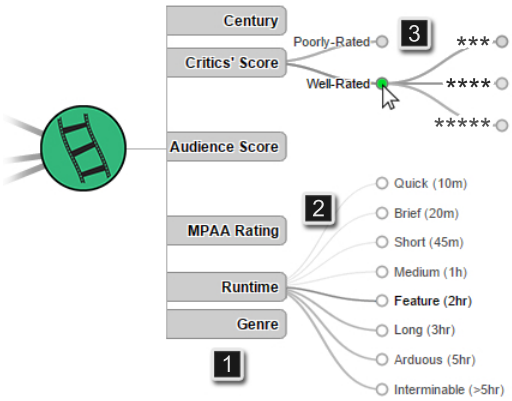


Figure 5: We visualize the nodes' feature space using a dendrogram view [10]. Hierarchical features can be expanded to reveal subsequent levels (3). The darker and thicker edges (2) are highlighted based on the results from the current query. Starred nodes (those whose value is fixed) highlight the node features in blue.

the query. She performs the search again with these fixed nodes. The query is now looking for any actor cast by the Coen brothers that was in *O' Brother Where Art Thou?* and any other Coen film. She receives the result, in Figure 2-3, showing George Clooney in *Intolerable Cruelty*. At any time she can remove the fixed nodes to return to a more abstract query.

Bernadette is curious if there are any movies from the 90's that fit her query. She right clicks the top film, reopening the feature selection dendrogram (Figure 5). She clicks 1990 and fetches new results. VISAGE allows her to quickly construct and iterate her queries.

3. VISAGE OVERVIEW

The VISAGE interface is comprised of a force-directed query visualization (Figure 4), a context menu that summarizes the node's feature conditions (Figure 4-2.1), a right-click, context-menu that summarizes features (Figure 4-2.2 in blue), a feature exploration pop-up panel (Figure 5), and a panel with a list of the current query results (Figure 4-2). As the analyst constructs their query, partial results are fetched in the background. User-selected nodes have a blue border and display a context menu (*O Brother, Where Art Thou?* in Figure 4-2).

If the analyst has added any feature conditions to the query, they will appear in green when a result node is selected (in Figure 4-2.1 the analyst has chosen only *Well-Rated* films). When a result is selected, a summary of the current node's features is shown in blue. If a particular node value from the data has been *starred*, its value in the query is fixed and can take only that specific value during the querying. Starred nodes have a golden star in the upper right and an additional context menu that reminds the user that the film is starred.

Adding new nodes is streamlined via our node tray, which is brought up by clicking the "+" icon on an existing node or right clicking on the background (see Figure 3.2). This menu displays the types of nodes that, if added, guarantee at least one match in the underlying network. The bars and percentages show the breakdown of neighbor types for the selected node. Each node shows a pin, a star and a magnifying glass when moused over. The pin spatially pins the node and the star allows users to keep it constant in the query. The magnifying glass opens the node search menu, in Figure 3.1, which allows users to search for particular nodes via text. Users can quickly and easily add known values and pin them; facilitating QBE-like query construction.

VISAGE Querying Language. Our demonstration of VISAGE will show the audience how to form graph queries, where the nodes can be as abstract as a wildcard (denoted as a "?" in the node type menu) or as constrained as taking a single value. Currently we support continuous, discrete, and categorical attributes; however, we do not support queries whose results are purely numeric as opposed to a collection of subgraphs.

Feature Guidance. VISAGE visualizes a summary of the current results' features by showing different branch thicknesses in the feature dendrogram (Figure 5.2). With knowledge of different attributes, analysts are better able to understand how the results' features are distributed.

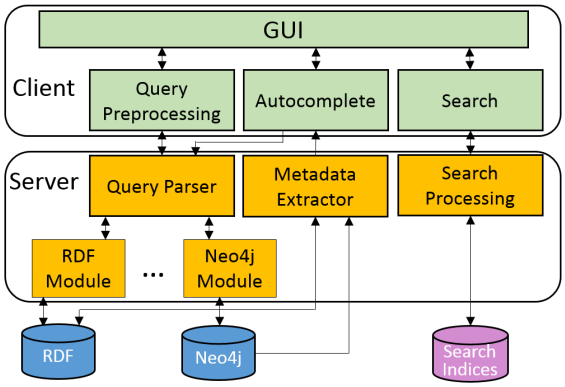


Figure 6: Visage utilizes a client-server architecture, where the client renders the user-interface in-browser (via javascript and HTML). The server is a lightweight python server which wraps graph databases (Neo4j, RDF; and potentially others). The *metadata extractor* creates summarization statistics for autocomplete.

Parsing A Graph Query. When seeking subgraph matches, if the starting node has very few matches in the graph, the search space is reduced and fewer comparisons are needed. Because the specificity of node feature conditions can vary from completely abstract (like a wildcard) to a single specific node, we have designed VISAGE to partition each query into a series of subqueries. We rank the nodes by the number and severity of their conditions. Starred nodes are parsed into subqueries first as their lookup is essentially constant time. VISAGE then ranks the remaining nodes by number of conditions. The entire parsing requires only a few milliseconds. For databases with strong query-optimization or those which do not support subqueries, this step can be skipped via a configuration done once before running VISAGE.

Implementation. VISAGE uses a client-server architecture (Figure 6) which separates the user-facing, interactive visualization (client) from the stored graph and a database management system (DBMS) to process it (server). The server utilizes modules to interact with multiple graph DBMSs, making VISAGE flexible across graph DBMSs. VISAGE presently can handle queries that map into a subset of Cypher (the querying language for Neo4j DBMS [6]). Additionally, it can also map queries into a subset of SPARQL, with full support in the near future. VISAGE's web client (Javascript and D3 [3]) and server (Python) can run smoothly on the same commodity computer, returning results like Figure 4 in less than a second (e.g., we developed VISAGE on a machine with Intel i5-4670K 3.65GHz CPU and 16GB RAM). Optionally, for larger graphs, the server may be run on a separate, more powerful machine.

4. CONCLUSION

We present VISAGE, an interactive visual graph querying approach that empowers analysts to construct expressive queries, without writing complex code. VISAGE allows the user to work gradually from abstract to specific example-driven queries through novel interaction techniques like *graph-autocomplete* that helps prevent over-specification. We will demonstrate and highlight VISAGE's features using a few usage scenarios on the popular Rotten Tomatoes film-actor-director graph, and we will invite our audience to try out VISAGE and freely experiment with their own queries.

5. ACKNOWLEDGMENTS

This research has been supported in part by NSF IGERT grant 1258425, NSF grants IIS-1563816, TWC-1526254, and IIS-1217559.

6. REFERENCES

[1] C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proceedings of the ACM SIGCHI Conference on Human factors in Computing Systems (CHI)*, pages 619–626, 1992.

[2] S. S. Bhowmick, B. Choi, and S. Zhou. Vogue: Towards a visual interaction-aware graph query processing framework. In *Proceedings of the Biennial Conference on Innovative Data Systems Research (CIDR)*, 2013.

[3] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[4] N. Cao, Y.-R. Lin, L. Li, and H. Tong. g-miner: Interactive visual group mining on multivariate graphs. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15. ACM, 2015.

[5] 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 *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pages 963–966, 2008.

[6] D. Montag. Understanding neo4j scalability. Technical report, Neo Technology, January 2013.

[7] S. Pandit, D. H. Chau, S. Wang, and C. Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th international conference on World Wide Web*, pages 201–210. ACM, 2007.

[8] R. Pienta, A. Tamersoy, A. Endert, S. Navathe, H. Tong, and D. H. Chau. Visage: Interactive visual graph querying. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, AVI '16, pages 272–279. ACM, 2016.

[9] R. Pienta, A. Tamersoy, H. Tong, and D. H. Chau. Mage: Matching approximate patterns in richly-attributed graphs. In *IEEE International Conference on Big Data (BigData)*, 2014.

[10] A. J. Saldanha. Java treeview-extensible visualization of microarray data. *Bioinformatics*, 20(17):3246–3248, 2004.

[11] A. Tamersoy, E. Khalil, B. Xie, S. L. Lenkey, B. R. Routledge, D. H. Chau, and S. B. Navathe. Large-scale insider trading analysis: patterns and discoveries. *Social Network Analysis and Mining*, 4(1):1–17, 2014.

[12] P. C. Wong, D. Haglin, D. Gillen, D. Chavarria, V. Castellana, C. Joslyn, A. Chappell, and S. Zhang. A visual analytics paradigm enabling trillion-edge graph exploration. In *Proc. LDAV. IEEE*, 2015.

[13] M. M. Zloof. Query-by-example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.