VIGOR: Interactive Visual Exploration of Graph Query Results

Robert Pienta, Fred Hohman, Alex Endert, Acar Tamersoy, Kevin Roundy, Chris Gates, Shamkant Navathe, Duen Horng Chau

Abstract—Finding patterns in graphs has become a vital challenge in many domains from biological systems, network security, to finance (e.g., finding money laundering rings of bankers and business owners). While there is significant interest in graph databases and querying techniques, less research has focused on helping analysts make sense of underlying patterns within a group of subgraph results. Visualizing graph query results is challenging, requiring effective summarization of a large number of subgraphs, each having potentially shared node-values, rich node features, and flexible structure across queries. We present VIGOR, a novel interactive visual analytics system, for exploring and making sense of query results. VIGOR uses multiple coordinated views, leveraging different data representations and organizations to streamline analysts sensemaking process. VIGOR contributes: (1) an exemplar-based interaction technique, where an analyst starts with a specific result and relaxes constraints to find other similar results or starts with only the structure (i.e., without node value constraints), and adds constraints to narrow in on specific results; and (2) a novel feature-aware subgraph result summarization. Through a collaboration with Symantec, we demonstrate how VIGOR helps tackle real-world problems through the discovery of security blindspots in a cybersecurity dataset with over 11,000 incidents. We also evaluate VIGOR with a within-subjects study, demonstrating VIGOR’s ease of use over a leading graph database management system, and its ability to help analysts understand their results at higher speed and make fewer errors.

Index Terms—graph querying, subgraph results, query result visualization

1 INTRODUCTION
Mining graph patterns, whether suspicious, anomalous, malicious, or just interesting, has become a critical technology for data analytics. For example, in financial transaction networks, analysts may want to flag “near cliques” formed among company insiders who carefully timed their activities [42]. Or in online auctions, analysts may want to uncover “near-bipartite cores” formed among fraudsters and their accomplices [29]. While there is significant research interest and development in graph algorithms, database management systems and even visual graph query construction techniques [2, 7, 32], much less work has focused on helping analysts make sense of the graph structure and rich data that makes up subgraph results. Visualizing graph query results (or matches) poses significant challenges, because we must effectively summarize: the underlying data from the nodes, the structure of each subgraph result, a large number of results, and the potential overlap in node and edges among results.

In this work, we visualize the resulting subgraphs from exact graph querying, in which the structure of nodes and edges matches exactly what the analyst specified in their query. Exact graph querying is used in many domains, from bioinformatics [39], cybersecurity [29], social
network analysis [21], to finance [42].

Most graph mining tasks are considered finished when query results have been returned; however, for analysts, seeing initial query results is only the beginning of their sensemaking process. Despite the significant interest in graph database management systems (DBMSs) and querying techniques, little investigation has been done in the space of graph query result visualization and exploration. Contemporary graph querying systems provide only basic methods for displaying results, often using tables or long lists (see examples in Figure 2). Given only the table and list visualizations, it’s a challenge to determine what groupings of similar results occur or how a particular node value appears among the results. In the current paradigm, analysts must first find patterns manually in a table before they can rewrite their original queries to do any filtering or grouping. This can be tedious and does not promote the development of an internal representation of the information space [36].

We present a novel visual analytics system, VIGOR, for exploring and making sense of graph querying results. VIGOR uses multiple coordinated views, leveraging different data representations and organizations to streamline analysts’ sensemaking process [18, 34]. The important contributions of VIGOR include:

- **Exemplar-based interactive exploration.** VIGOR simultaneously supports **bottom-up** sensemaking [36], where an analyst starts with a specific result and relaxes constraints to find other similar results; and **top-down** sensemaking, where the analyst start with only the structure (i.e., without node value constraints), and add constraints to narrow in on specific results (Figure 1A). VIGOR supports analysts when investigating how many values are matched to each query-node and how a particular node value filters the results.

- **Novel result summarization through feature-aware subgraph result embedding and clustering.** VIGOR provides analysts with a **top-down**, high-level overview of all their results which enables analysts to handle complex grouping and comparison tasks to make sense of their data [28, 36]. We introduce an algorithm to group results by node-feature and structural result similarity (Figure 1C) and embed them in a low dimensional representation. By grouping similar results into clusters and making cluster comparison easy, analysts can quickly detect and understand underlying patterns across their results.

- **An integrated system fusing multiple coordinated views.** VIGOR provides multiple brushable linked views to flexibly explore and make sense of subgraph results, by integrating the **Exemplar View**, **Subgraph Embedding View**, and the **Fusion Graph**. The **Fusion Graph** (Figure 1B) shows the subgraph from the underlying network created from combining all the results, in which very common or uncommon nodes will have high and low degree respectively. The coordinated views make it easier to see how nodes appear together across the many subgraph results.

- **Real world application to discover cybersecurity blindspots; advancing the state of the art** Through a collaboration with cybersecurity researchers at Symantec, a leading security company, we present the investigative analysis performed in and insights gleaned from using VIGOR to discovering and understanding blindspots in a cybersecurity dataset with over 11,000 real incidents. Through a usability evaluation using real co-authorship network data obtained from DBLP 1, we demonstrate VIGOR’s ease of use over Neo4J, a leading graph DBMS, and its ability to help users understand their results at higher speed and with fewer errors.

2 Introducing VIGOR

To illustrate how VIGOR works in practice, we will briefly cover an overview of the system’s components (in Section 2.1) and an illustrative scenario where we explore co-authorship in a DBLP network.

1DBLP Website: http://dblp.uni-trier.de/
Fig. 4. The Subgraph Embedding provides an overview of the results through the feature-aware subgraph embedding, where results are displayed as points in two dimensions based on node feature similarity. We see the clustered results of a query seeking two co-authors of two papers at VAST and another conference (shown in Figure 3). Nearby clusters (A) and (B) both contain VAST and KDD papers, the features of which are compared in Figure 6. Cluster labels are customized by the analyst during exploration.

To VAST and another conference. In VIGOR, Alexis starts by entering a query written in the Cypher query language from the popular Neo4j (http://neo4j.com) DBMS. Her query appears graphically in our Exemplar View, where she verifies that she correctly specified the right structure (Figure 3A).

Alexis has just begun her investigation and she wants to see an overview of her results. She gets over 2,500 results, each with six nodes (from the previously mentioned query), wherein some nodes could be shared among multiple results. She wants a high level overview of her results that allows her to see similarities and groupings.

VIGOR’s Subgraph Embedding view provides an overview of all her results in the form of a plot with clusters. Similar subgraph results (gray squares in Figure 4) are placed spatially close together by the feature-aware subgraph result embedding and clustering (Section 4). To help her differentiate among groups, VIGOR uses a density-based clustering technique [22] to detect clusters and automatically creates colored concave hulls for each. Alexis has the option to adjust the embedding and clustering parameters. She may also create and name her own clusters by lassoing groups of points (Figure 5A-C).

She shift-right-clicks two neighboring clusters (Figure 4A and 4B) to compare them in the Feature Explorer (Figure 6). The Feature Explorer shows common node values and feature distributions for each node type included in the clusters, similar to [35, 41]. The color of the plots in the Feature Explorer correspond to the colors of the selected clusters. She can use the value-plots (bar charts in Figure 6) to see what nodes appear most commonly in a cluster. Alexis labels the clusters based on their most common conferences (e.g., “VAST & UIST” in Figure 4). She notices that both clusters are composed of authors and their publications at VAST and KDD, a top tier data mining conference.

From the author feature distributions in the Feature Explorer (Figure 6-left) she discovers that the gray cluster (cluster B) is likely to contain more senior researchers, because they have higher paper counts, more distinct conferences and greater numbers of co-authors. Her curiosity grows as she wonders what types of papers bridge these two research communities.

After her initial query, Alexis is faced with numerous results, but she wants to find specific authors and papers. What should she do next? She can quickly filter down results by values with which she’s comfortable by clicking one of the yellow conference nodes in the Exemplar View window, which displays a selectable dropdown menu with the matching conferences (Figure 3B). She selects KDD. When she clicks on an author node in Figure 3A, the dropdown now contains only those authors who have published together at VAST and KDD. From the list she recognizes Shixia Liu, a VAST’17 Paper Chair, and selects her (Figure 3C).

Alexis’ selection in the Exemplar View filters the Fusion Graph (Figure 7), a force-directed graph induced by joining all subgraph matches together (e.g., if a conference is shared among several results, it will appear only once). The Fusion Graph now shows only Shixia Liu’s co-authors on at least one paper with her from VAST and one from KDD (e.g., Michelle, Furu, Li, Xiting, and Baining in Figure 7). Alexis discovers that each paper is related to understanding textual data and is potentially valuable to her future research. She is inspired by the combination of the speed, scale, and automation of data mining being combined with the visual, interaction design, and sensemaking of visualization.

Fig. 5. (A) Starting from a group of results, (B) an analyst lassos the desired results. (C) A concave hull is established forming a cluster with the points. Cluster can be used to: filter the Fusion Graph and compare features and node values in the Feature Explorer.

Fig. 7. Shixia Liu’s papers and co-authors who have published papers together at VAST and KDD. The Fusion Graph view shows an induced subgraph of all the combined results from the original query, which can be filtered from either the Subgraph Embedding or the Exemplar View.
3 **Core Design Rationale**

Below, we present the core facets of VIGOR’s design and discuss how they support sensemaking for query results.

### 3.1 Leveraging Examples: Bottom-Up Exploration

Starting with low-level details is often referred to as a bottom-up sensemaking [31, 36]. Starting from a known example can greatly improve the development and understanding of a query [49]. We designed the Exemplar View (Figure 3) to provide the following: (1) an arrangeable visualization of the typed input query for fast error-checking; (2) easily accessible information on how many values a particular node from the query finds in the results (e.g., does an author node in a query match to only 3 authors or 3,000?); (3) the ability to start from a familiar result and relax constraints to find other results; and (4) a fast mechanism to add node value constraints to filter down the number of results.

At every step of relaxation in (2) or filtering in (3), the analyst sees real-time updates (in dropdowns in the Exemplar View and as filtering in the Fusion Graph) as the number of possible results changes. Conversely, if the analyst adds new node value constraints

### 3.2 A View From Above: Top-Down Exploration

High level overviews, like the Subgraph Embedding (see Figure 4), have proven useful in visualization models for sensemaking in other datasets [28, 31]. An overview of subgraph results is challenging, because: the number of subgraphs is large, the subgraphs may share nodes and edges, and each subgraph is made of multiple nodes that each have separate (and often very different) features.

To overcome these challenges we represent each result as a square glyph (to differentiate from the circles used for nodes) rather than nodes and edges, to simplify plotting. The Subgraph Embedding has the strengths of a scatterplot (including concave hulls around clusters) of all the results based on their nodes’ features. The Subgraph Embedding allows zooming, panning, jitter, and fine-grain control over embedding and clustering. We group similar results with concave hulls, because there are many cases in which convex hulls overlap unnecessarily. New clusters can be freely created using a freeform lasso tool. Similar results are plotted close to each other and often form clusters as in [45].

The details of our graph embedding algorithm are discussed further in Section 4.

### 3.3 Feature-centric Sensemaking for Result Clusters

Typically, when an analyst poses a query they have constrained only some of the potential features of their results; the remaining features are free to vary and often form patterns. Feature distributions [41] and node-feature distributions [35] have proven a valuable way to compare results. To compare these features, we created the Feature Explorer (Figure 6), which provides node feature and value distributions by node type for a cluster. The lasso can be used to create new clusters, even from within other clusters or combining them. Multiple clusters can be compared at once by selecting them in the Subgraph Embedding.

### 3.4 Coordination in Multiple Views

VIGOR utilizes linked highlighting and filtering so that changes made in one view are reflected in the others. The Exemplar View highlights the Subgraph Embedding and filters or highlights the Fusion Graph based on node-value constraints. Clicking squares or clusters in the Subgraph Embedding: allows the selection an exemplar result in Exemplar View for bottom-up exploration, filtering or highlighting the Fusion Graph, and allows for the selection of different clusters in the Feature Explorer. Hover over a node in the Fusion Graph: highlights the node’s neighbors and the results containing that node in the Subgraph Embedding. An analyst can choose to filter or highlight the Fusion Graph with the Exemplar View and Subgraph Embedding, with filtering the default.

### 4 Methodology & Architecture

In the following section we outline our novel feature-aware, subgraph-result embedding for reducing subgraph-results to 2D points. While dimensionality reduction is common in other areas of visualization, visualizing graph query results has seen significantly less advancement. Dimensionally reducing subgraphs requires: (1) a graph embedding to turn each subgraph into a high-dimensional vector and (2) distance-preserving reduction techniques to reduce the dimensionality of each subgraph, without losing underlying similarities. We combine both structural features from the network topology as well as features from the nodes. Often some nodes may have missing values or different types making

#### 4.1 Embedding Subgraphs

For our embedding, we utilize both network topology features as well as the rich domain features from our nodes. The embedding pipeline takes four stages from result set to low-dimensional representation. The steps of the pipeline are (see Figure 8):

- **Extract Features** - Calculate the topological- and node-features.
- **Vectorize** - Merge the common features into per-result vectors.
- **Aggregate & Normalize into Signature** - Reduce the large input vectors into uniform signatures.
- **Reduce & Cluster** - Reduce the signatures using dimensionality reduction to fit them into 2D.

Our Subgraph Embedding reduces query results (each is a subgraph) into points via a subgraph embedding for visual results similar to [45]; however, our approach differs in several key areas outlined below.

#### Extract Features

We use both the node-features $f_i$ and a small set of topologically extracted features $f_j$ as inputs to our embedding (Figure 8A and 8B). There are many different ways to extract features from a graph. We started with the structural features from [45] and NetSimile, [4], for structural features. Based on our experiments using structural features alone is insufficient in our case. Often our subgraph results have significantly fewer nodes than both previous approaches and have exactly the same network structure. Because of the identical structure of our subgraphs the embedding from [4] will project all the results into a single point.

We integrate some of the novel features from NetSimile, but leave several out as they did not perform well on our induced subgraphs. Unlike both approaches we make use of the node features from the results themselves in our embedding. This means that different nodes with similar features will be closer to each other, increasing the chances of semantically meaningful and explainable clusters. In the case of real-world data nodes may be missing values, which makes a purely feature-driven comparison between results imbalanced (as some results may have features that others do not). We address this problem by converting the raw features to fixed-length signatures, which we cover in the **Aggregate & Normalize into Signatures** subsection. Which node features to use are chosen by the analyst in a network schema configuration done once during VIGOR setup.

Assume we have received $k$ results, where each result is composed of $n$ nodes. For just the structural features we look at each result in the context of the original network and extract subgraph neighborhood and egonet information from the underlying graph. An egonet of a node, $i$, is the neighbors of $i$, the edges to these neighbors and all the edges among neighbors. This performs significantly better for small queries by structurally differentiating them based on their place in the underlying data. The most effective structural features are:

- **Node degree** - or the number of neighbors $d_i = |N(i)|$

  where $N(i)$ is the set of neighboring nodes of node $i$.

- **Egonet edges** - the sum of inter-neighbor edges of node $i$

  $$E(ego(i)) = \sum_{j \in N(i)} \left( \sum_{e_{k \in E(j)}} \delta_{ik} \right),$$

  $$\delta(ik) = \begin{cases} 1 & \text{if } k \in N(i) \\ 0 & \text{if } k \notin N(i) \end{cases}$$

  where $e_{j,k} \in E(j)$ are the edges at node $j$ to node $k$. 

Where $E(ego(i))$ is the sum of edges between node $i$ and its neighbors.
We use the first 4 moments: mean, variance, skewness, and kurtosis. For frequency distribution functions (mean, variance, skewness, and kurtosis), which collapse the original sub-vectors of different lengths into new uniform-length vectors (D), each is unraveled into a signature for the result, which are unit-normalized and dimensionally reduced (E). The low-dimensional space is clustered before the results are presented (E).

**Vectorize**

Each node of a result now has the four structural features as above and any non-text features from the nodes themselves (e.g., for an author node in our DBLP graph, we have additional features like the number of coauthors, number of conferences, etc.). This creates an issue, both because a result has k different feature vectors and also the different types of nodes in the result will have different lengths of features (see Figure 8C). The first problem we solve by vectorizing the features per result. We merge common features across the nodes into a single vector per feature for each result (Figure 8B).

**Aggregate & Normalize**

To solve the issue of uneven lengths of the per-result feature vectors we convert them into a signature (see Figure 8D). We aggregate each vector down to a fixed number of values such that the signatures are all the same length. We utilize the moments of distribution functions (mean, variance, skewness, and kurtosis) which collapse the original sub-vectors of different lengths into new uniform-length vectors (D), each is unraveled into a signature for the result, which are unit-normalized and dimensionally reduced (E). The low-dimensional space is clustered before the results are presented (E).

• **Egonet neighboring nodes** - the total number of neighbors across all the neighbors

\[
|N(ego(i))| = \bigcup_{j \in N(i)} N(j)
\]

• **Clustering coefficient** - the fraction of closed triples over total triples from the neighbors of node i

\[
c_i = \frac{2|\{j, k \in N(i) : e_{j,k} \in E(i)\}|}{|N(i)| \cdot (|N(i)| - 1)}
\]

**Reduction & Cluster**

We perform clustering on the dimensionally reduced points (see Figure 8E). There are many density-based clustering algorithms like DBSCAN [37] or OPTICS [22]. We use OPTICS to perform our density-based clustering, because it performs better on clusters with different densities [22]. Because the choice of \(\epsilon\) greatly affects the resulting clusters, we allow the user to adjust the value via a slider. The cluster information is encoded as colors in the Subgraph Embedding.

4.2 Architecture

VIGOR uses a client-server architecture using D3 and jQuery for the front-end and python for the back-end. The network data are stored using the popular Neo4j graph database. We chose Neo4j for its cross-platform support, robust querying language, and its scalability to large graph datasets. One of our goals is to offer VIGOR as a flexible sensemaking tool that works on a wide variety of network datasets. Our design separates the underlying network schema from the system, so that VIGOR can easily be used on different network data.

**Performance**

VIGOR is a practical working prototype analytical system; the queries shown in this paper are all returned within 1-2 seconds. We achieve this performance through Neo4j indices and asynchronous computation of dimensionality reductions. Because the different dimensionality reduction techniques have significantly different run times, we return PCA (the fastest) first to maintain the interactivity of the system and subsequently return the others in the background.
5 Evaluation

We performed a two-part user evaluation of VIGOR (Section 5.1). In the first part, we compare VIGOR against Neo4j, a leading graph DBMS. Neo4j is an industry leader among the few free systems that visualize graph query results. In the second part, we performed a think-aloud investigation of the Subgraph Embedding, because there is no analog in Neo4j against which to compare.

To study how VIGOR can help with solving real-world problems, we collaborated with three security researchers at Symantec\(^1\), the leading security company, to identify blindspots in the understanding of critical security incidents. In Section 5.2, we present the investigative analysis performed and insights gleaned from using VIGOR on an cybersecurity incident-network.

The details of the analyzed graphs are outlined in Table 1.

<table>
<thead>
<tr>
<th>Network Type</th>
<th>Node Edges</th>
<th>Node Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBLP</td>
<td>115,989</td>
<td>1,543,792</td>
</tr>
<tr>
<td>Cybersecurity</td>
<td>17,651</td>
<td>384,172</td>
</tr>
</tbody>
</table>

Table 1. Graph datasets used for evaluation: DBLP dataset for user study; cybersecurity dataset for real-world application to discover security blindspots.

5.1 User Study

To evaluate VIGOR, we conducted a user study to assess how well our new visualization techniques compare to the current state-of-the-art Neo4j interface. Previous research has focused on how analysts can visually construct queries \([7, 32, 49]\); however, our research focuses on how well analysts can make sense of and solve tasks given a set of query results. We chose a DBLP co-authorship graph, because the concepts are relatively simple and accessible to non-expert participants.

Our protocol has two parts: (I) comparative tasks, (II) a think-aloud exploration study. In Part I, we measured the number of errors and time taken solving a set of tasks for both VIGOR and Neo4j. In Part II, we asked participants to perform some open-ended exploration objectives after giving them a tutorial on the think-aloud protocol.

5.1.1 Participant Demographics

We recruited a total of 12 participants via our institutions local mailing lists. They ranged in age from 21 to 31, with 25 as the average. Of the participants, 7 were female, while the rest were male. Each study lasted on average 70 minutes, for which the participants were each paid $10 for their time.

5.1.2 Protocol

We utilized a within-subjects experimental design with two systems (VIGOR and Neo4j) and two task sets. Each system was tested with one of two sets of tasks (see the subsequent Task section). Participants completed the first set of tasks with the first system and the remaining task set with the second system. System order was counterbalanced to ensure experimental fairness. Task sets were also counterbalanced for fairness.

Participants were given an introduction to the dataset and tutorials of each system before being given the tasks. We encouraged participants to ask questions at any time during the study, but especially during the introductory period. For Neo4j we created an interactive Neo4j tutorial tailored to our dataset and instructed participants on Neo4j’s interface and its features. For VIGOR we provided an interactive tutorial of the interface, how to filter results, and how to interact with our views.

Once a participant had completed tutorial for their current system, we provided them with context in the form of a scenario based around each query; participants were not asked to write queries. We then instructed them to work quickly and accurately on each task. Each task was allotted five minutes and was timed separately. Participants could only move onto the next task once they had completed the current task.

5.1.3 Part I: Comparative Study

Tasks

Our interest was in testing the speed of solving simple tasks with a collection of results rather than the speed of writing queries. For each task the participant was provided a short scenario and a pre-written query. The patterns for the tasks were based on common patterns and motifs from prior graph mining research \([14, 21, 33]\). The tasks from Task Set A (shown in Figure 9) are:

1. Find the count of ICDM conference papers by Daniel Keim in our dataset.
2. From the last two years of KDD publications, find and list the authors who are on more than one paper with “entity” in the name.
3. Find the number of distinct groups of researchers that Tobias Shreck is in from INFOVIS publications.
4. Among coauthors of at least two papers together at INFOVIS and KDD, who has the most publications.

5. Among coauthors of at least two papers together at INFOVIS and KDD, who has the most publications.

The tasks approximately increased in difficulty from 1 through 5. We ranked the difficulty of each task based on the number of nodes, edges, complexity of the query, and size of the results. Our initial intuition was that Neo4j and VIGOR would achieve similar performance for the easier early tasks, while VIGOR would be faster for harder queries.

Error rate and task completion time were the dependent measures. Both measures could be affected by: (1) Software (VIGOR or Neo4j); (2) Task Set (Set A or Set B); (3) Software Order (VIGOR or Neo4j going first). Because of the within-subjects design we utilized a Latin Square design randomizing each participant into one of four groups where we counterbalanced the possible confounding factors (e.g., one group is (VIGOR + Task Set A) then (Neo4j + Task Set B)).

Fig. 9. VIGOR user study comparative tasks. These tasks were provided to create the result sets used in Part I of our user study. Both task sets utilized the same query topologies, but different values, carefully selected to have the same number of results.

\(^1\)We invited our Symantec collaborators to join as coauthors of this work.
After the comparative tasks were completed, all participants were asked (Figure 11). One participants stated, “I enjoyed the clustering features (easier to learn, easier to use, and more likeable overall; although this is using VIGOR more than a Neo4j and reported that our system was: about any possible combinations of groups.” The participants enjoyed of VIGOR , allowing the user to quickly compare variables (Year, etc.) participants felt that VIGOR was better than Neo4j for all 7 aspects asked to rate various aspects comparing both systems using Likert scales. Par-
than in Task Set B. Participants were both significantly faster and less paring the results easy (Part II).

5.1.5 Discussion and Limitations

The qualitative and quantitative results of our user study were positive. The results suggest that VIGOR provides useful and effective visual techniques for analyzing and making sense of graph query results. VIGOR achieves this improved performance through: (1) streamlining the filtering process to allow users to quickly narrow down by a particular author (Task 3), or by a particular term in papers (Task 2); (2) the flexibility and customization of the Fusion Graph graph layout (all Tasks); and (3) the Subgraph Embedding, which makes grouping and comparing the results easy (Part II).

While Neo4j is an industry leader, we found two specific design-choices (based on participant feedback) that limited performance with Neo4j: (1) the default edge-autocomplete, add any underlying edge from the network (regardless of its inclusion in the query); and (2) the instability of the force directed layout positions during node dragging.

We did not evaluate query creation and modification. Our study did not evaluate query creation and refinement; participants were given the query that corresponds to a scenario investigating co-authorship, which may not be the most natural query that they would like to create. If we allowed participants to create ad hoc task queries, the immense variety of possible queries would make the evaluation extremely difficult. Moreover, query refinement, a challenge that would add additional confounding factors to the study, would also require participants to have more prior knowledge [32]. Even the queries provided were challenging to many participants, as demonstrated by the high error rates in.
We collaborated with three security researchers from Symantec to identify blindspots in their company’s understanding of critical security incidents. They see strong potential in VIGOR to help them educate their company customers about these weak points in their response to dangerous security situations. We used VIGOR in the following ways to identify company blindspots and bring them into focus: (1) we contrasted companies that tend to ignore critical security incidents with peers that face the same types of incidents but exhibit exemplary incident response (see Figure 12A), and (2) we highlighted instances in which companies do not respond consistently to critical security incidents such as vulnerability scans and malware outbreaks (see Figure 12B).

Symantec Cybersecurity Network To pose these types of queries, we created a cybersecurity network composed of Company nodes (orange), Incident nodes (red) and Signature nodes (blue). In total, this network of security data contains 17,651 nodes and 384,182 edges. All of these entities correspond to real world events and represent the detection of and actions taken against various security threats. Companies are linked to the security incidents that were detected on their systems, and each incident is in turn linked to the signatures that were responsible for triggering it. Signatures that are responsible for the creation of security incidents are designated as “Active Signatures” that typically identify glaring security issues, such as malicious network traffic and computer viruses. Security products also define “Passive Signatures”, whose primary purpose is to provide contextual information about such things as login behavior and other system or network events. The active signatures trigger security incidents of various levels of severity, of which critical incidents are the most important, and are the basis of the incidents used in case study, as they should be met with exemplary response to dangerous security situations.

Query 1 - Comparing Company Incident Behavior Our first query (see Figure 12A) identifies pairs of companies faced with critical security incidents that consist of at least one active and one passive signature, such that one company resolved its incident while the other company ignored it. By posing this VIGOR query and examining its results (ref. Figure 13), we identify a company’s (Company 7) blindspots in a way that simultaneously provides interactive graphical evidence that another company (Company 16) is faced with similar security incidents and takes them seriously. The results of this query can also be used as an educational tool and shared directly with companies as evidence of their most glaring blindspots. Doing so helps companies re-evaluate and react differently to future instances of incidents that they would have otherwise continued to ignore.

Query 2 - Inconsistent Company Threat-Reactions Of similarly concern are situations in which a company reacts inconsistently to the same type of security incident. By posing the query of Figure 12B, we identify incidents that companies respond to inconsistently. This query provides a company with the ability to identify blindspots at the finer-grained level of its individual incident responders, some of which may understand the perils of a particular type of security incident much better than others do. Example results are shown in Figures 14A and 14B. In both cases, VIGOR identifies malware outbreaks that were not fully eradicated. Further outbreaks of the malware are likely in both cases. The malware of Figure 14A could be spreading by means of the unmitigated unauthorized internal vulnerability scans that happened in a similar timeframe. Similarly, internal machines still infected by the trojan malware of Figure 14B could be used by an attacker to re-establish a firm foothold within the targeted company since the compromised machines were not all cleaned. An additional benefit of this VIGOR query and visualization is that it functions very well as a progress checker after a major security issue, allowing companies to track their progress as they work to ensure that malware outbreak are fully eradicated from the environment. Finally, Figure 14A and 14B both highlight the way in which the Feature Embedding is able to cluster related security blindspots in two dimensions for efficient perusal.

6 RELATED WORK

Graph Visualization and Query Languages Visualizing graphs is a challenging topic that has attracted significant research interest and motivated the development of many tools and techniques. Herman et al. [16] conducted an extensive survey of much of the foundational work. More recently developed techniques for static graph visualization have been covered in [47] and for dynamic graphs in [3]. Graph sensemaking and interaction have grown in popularity [31]. Our work extends this large body of research by providing a visual approach to graph query result understanding and exploration.

Query By Example [49], is an early bottom-up querying system allows users to formulate queries by creating templates from “example queries” rather than writing conventional SQL statements. Another key innovation is to abstract the exact underlying data schema away from analysts as in PICASSO [20], which uses visual glyphs to create visual database queries. Both [1] and [40] avoid complex data schemas.
in favor of graphical widgets. For a further detail of visual querying languages on relational databases see [8]. Data storage techniques like the extensible markup language (XML) and resource description framework (RDF) have spurred other querying languages like XQUERY [6], XPATH [12], SPARQL [30]. Both [9] and [27] propose graphical querying languages for XML, while Hogenboom et al propose one for RDF data [17]. Our work builds on visual querying by using visual metaphors for both the query and the results.

Graph Querying and Graph Databases Algorithmically determining if a given subgraph exists within another graph is referred to as the subgraph isomorphism problem and is \textit{NP-Complete} [13]. Tong et al. proposed to use random walk with restart probabilities to heuristically score approximate matches in G-Ray [44]. The MAGE algorithm follows a similar heuristic and supports a much wider set of input queries [33]. Tian et al. utilize approximation indices to provide real-time approximations to a user specified query [43]. While we use Neo4j as our underlying database, most graph matching frameworks work with the ideas we propose in this paper.

Visual Graph Querying There are a few recent visual graph querying systems, many focus on the construction of queries rather than the presentation of results. GRAPHITE [11], allows users to visually construct a graph query over categorically attributed graphs. VOGUE [5], is a query processing system with a visual interface that interleaves visual query construction and processing. Cao et al. created g-Miner, an interactive multivariate graph mining tool that supports template matching and pattern querying [7]. VISAGE [32], is another graph querying tool which guides the user using graph-autocomplete, or pre-fetched results to help guide analysts towards results. GRAPHITE, VOGUE, and VISAGE all use lists to present the results to the user and focus considerably less on the result visualization than the query formulation. Our work aims to fulfill this gap in the current research, by proposing new methods to summarize and explore graph query results.

Summarizing Graphs, Kernels and Embeddings Another line of research focuses on “summarizing” graphs. Koutra et al. [21] propose VoG, which constructs a vocabulary of subgraph-types like stars and cliques to simplify visualization. Dunne and Shneiderman [14] present motif simplification, wherein common patterns or motifs are replaced with easily understandable glyphs (e.g. fans and cliques), which was subsequently applied to biological networks in MAVisio [39]. Rather than replacing structural elements, graph kernels and embeddings allow graphs to be converted in the vectors and scalars. There are numerous types of graph kernels and kernel similarity methods [46]. Both [48] and [46] use the structure to create the embedding while NetSimile, [4], uses extracted features. Van den Elzen et al. used graph embedding to plot the changes in dynamic graph snapshots over time [45]. We draw on some of these ideas in our Embedded Results view to collapse each result down to a point, the details of which will be discussed in the Methodology & Architecture section.

7 Discussion and Future Work

When implementing visual graph querying systems, we must grapple with two different scalability concerns: the visual and the computational. The visual scalability of our system is primarily limited by the Fusion Graph, which quickly accumulates large numbers of nodes and edges. By using the Exemplar View, and the Subgraph Embedding, analysts can quickly filter down the Fusion Graph to manageable sizes. The computational scalability of our model is most limited by the dimensionality reduction techniques like t-SNE and MDS, while PCA and kernel-PCA run in under a second. The time to fetch the query results was often trivial compared to the time needed for the embedding pipeline.

We offer several forms of dimensionality reduction, because dimensionality reduction is challenging and the best solution often depends on the underlying data. The choice of which dimensionality reduction method as well as the parameters (ε and n_{nigh}) for OPTICS clustering have been left up to the user. Theses choices vary greatly with the underlying characteristics of the network data and suggest that the best options should come from collaboration between a visualization expert and a domain expert. In our experience, the nonlinear dimensionality reduction techniques worked much better for clustering on most graphs; however, the axes of these approaches are much harder to interpret. Both t-SNE and MDS do a better job at preserving the small distances between the high dimensional points than conventional PCA and this likely leads to better clustering performance. VIGOR might benefit from an approach that automatically detects the dimensionality with the best clustering.

Currently VIGOR applied our system to exact subgraph matches; however, new systems may also produce approximate subgraph matches. Because the approximate results are not identical in shape and content, the result set becomes much more complex. Additional visualization techniques are needed to show where and how approximate results do not match the original query.

8 Conclusions

Visualizing graph query results is challenging, requiring effective summarization of a large number of overlapping subgraph results, each having complex network structure and rich node features. We presented VIGOR, a novel visual analytics system for exploring and understanding graph querying results.

VIGOR supports top-down and bottom-up result sensemaking, through its (1) exemplar-based interaction technique, where an analyst starts with a specific result and relaxes constraints to find other similar results or starts with only the structure (i.e., without node value constraints), and adds constraints to narrow in on specific results; and (2) a novel feature-aware subgraph result summarization. Through our collaboration with Symantec, we demonstrated how VIGOR helps discover security blindspots in a cybersecurity dataset with over 11,000 incidents. We also evaluate VIGOR with a within-subjects study, demonstrating VIGOR’s ease of use over a leading graph database management system, and its ability to help analysts understand their results at higher speed and make fewer errors.

Acknowledgments

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


[40] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borg-

