ADAPTIVE VISUAL NETWORK ANALYTICS: ALGORITHMS, INTERFACES, AND SYSTEMS FOR EXPLORATION AND QUERYING

A Dissertation
Presented to
The Academic Faculty

By

Robert S. Pienta

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computational Science and Engineering

Georgia Institute of Technology
December 2017

This research is supported by: National Science Foundation grants DARPA-ADAMS W911NF-11-C-0088, NSF IGERT grant 1258425 (FLAMEL), NSF grants IIS-1563816, TWC-1526254, and IIS-1217559. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entities

ADAPTIVE VISUAL NETWORK ANALYTICS: ALGORITHMS, INTERFACES, AND SYSTEMS FOR EXPLORATION AND QUERYING

Approved by:

Dr. Duen Horng Chau, Advisor
Computational Science and Engineering
Georgia Institute of Technology

Dr. Shamkant Navathe
Computer Science
Georgia Institute of Technology

Dr. Alex Endert
Interactive Computing
Georgia Institute of Technology

Dr. Bistra Dilkina
Computational Science and Engineering
Georgia Institute of Technology

Dr. James Abello
DIMACS
Rutgers University

Dr. Jilles Vreeken
Databases and Information Systems
Max Planck Institute of Information and Saarland University

Dr. Hanghang Tong
School of Computing
Arizona State University

Date Approved: June 27, 2017

Keywords: Graph Mining, Data Mining, Machine Learning, Sensemaking, Visualization, Visual Analytics, Graph Querying, Approximate Subgraph Matching, Graph Autocomplete, Visual Graph Query Construction, Visual Graph Query Refinement, Cypher, Neo4j, Subgraph Result Visualization, Graph Databases, Symantec Cybersecurity Blindspot Detection, FACETS, MAGE, VISAGE, VIGOR

There’s nothing funnier than a pointless epigraph.

Anonymous

For my parents, brothers, and sister.

ACKNOWLEDGEMENTS

My time at Georgia Tech has been an amazing experience. Polo, I am greatly indebted to you, for all the late night edits, weekend meetings, guidance, and long hours you committed to me and my research. Our first projects together steered the development of this dissertation and kept me on track. Your patience, eye for design, detail, and novelty was instrumental in the success of my research here at Georgia Tech. I am so grateful to have been a founding member of the Polo Club of Data Science. Thank you!

Alex, much of my research would never have happened without your advice, ideas, and your positive attitude (and jokes!). Before we met I knew little about the visual analytics research process, your guidance on VISAGE and VIGOR greatly improved the way I approach new visual analytics problems.

I would like to thank my thesis committee members, Shamkant Navathe, Alex Endert, Bistra Dilkina, James Abello, Jilles Vreeken, and Hanghang Tong for your advice, comments, and help making this dissertation possible.

Richard Fujimoto, your wisdom and guidance brought me to Georgia Tech, and introduced me to the FLAMEL program, from which I learned valuable interdisciplinary research and communication skills. To my FLAMEL fellows, you have opened my eyes to how challenging experimental work is and to how critical interdisciplinary research is to progress. Perry and Nils, your knowledge, drive, great humor, and our camaraderie were essential to my success.

To my academic siblings in Polo Club (past and present), y'all make Polo Club a great place to do research and collaborate: Acar Tamersoy, Chad Stolper, Lyés Khalil, Minsuk Kahng, Shang-Tse Chen, Nilaksh Das, Zhiyuan Lin, Peter Pollack, Yiqi Chen, Dezhi Fang, Nathan Dass, Paras Jain, Matthew Keezer, Jake Williams, and Kshitij Kulkarni.

The challenges of a Ph.D. are numerous and I would not have made it without my compatriots in the Crow Squadron, my cyclist gang. Thanks Yacin Nadji, Casey Battaglino,

and Taoran Le for helping me weather the storm. Casey and Yacin, I met you when I visited Georgia Tech, you played a major role in my coming to Georgia Tech and succeeding here. CAW!

Jordi, as a fellow BBQ-Boy and Research Goblin, you always kept morale high for our late night research sessions. I'll miss getting coffee and our *deep academic discussions*. Fred, your sincerity and meticulous eye for design has completely convinced me that Polo Club will not only survive, but thrive! I appreciate all your help with the design of and writing for several of my final thesis projects.

Last but not least, many thanks are due to my parents, brothers, and sister. You were supportive and enthusiastic throughout the process and I love you all.

Thank you all! Live long and prosper.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Large graphs are now commonplace, amplifying the fundamental challenges of exploring, navigating, and understanding massive data. Our work tackles critical aspects of graph sensemaking, to create human-in-the-loop network exploration tools. This dissertation is comprised of three research thrusts, in which we combine techniques from data mining, visual analytics, and graph databases to create scalable, adaptive, interaction-driven graph sensemaking tools.

(1) **Adaptive Local Graph Exploration**: our FACETS system introduces an adaptive exploration paradigm for large graphs to guide user towards interesting and surprising content, based on a novel measurement of surprise and subjective user interest using feature-entropy and the Jensen-Shannon divergence.

(2) **Interactive Graph Querying**: VISAGE empowers analysts to create and refine queries in a visual, interactive environment, without having to write in a graph querying language, outperforming conventional query writing and refinement. Our MAGE algorithm locates high quality approximate subgraph matches and scales to large graphs.

(3) **Summarizing Subgraph Discovery**: we introduce VIGOR, a novel system for summarizing graph querying results, providing practical tools and addressing research challenges in interpreting, grouping, comparing, and exploring querying results.

This dissertation contributes to *visual analytics*, *data mining*, and their *intersection* through: interactive *systems* and scalable algorithms; new *measures* for ranking content; and *exploration paradigms* that overcome fundamental challenges in visual analytics. Our contributions work synergistically by utilizing the strengths of visual analytics and graph data mining together to forward graph analytics.

CHAPTER 1
INTRODUCTION

The volume of information available today is enormous and constantly increasing. People use data of all kinds to make sense of the world. Insight extracted from data are essential for planning, decision making, and understanding our lives. Progress in education, science, and technology requires us to make sense of and gain insight from overwhelming quantities of data. From a user's or analyst's (we will use them interchangeably) perspective, the main challenge of sensemaking lies not in storage, or computational efficiency, or even large scale data processing. It lies in how to best augment an analyst's limited cognitive faculties to make sense of a large data corpus.

Significant amounts of data can be captured and stored as a graph; capturing the complex relationships among entities. Understanding networks 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). The focus of this dissertation is on improving sensemaking process for graphs or networks. Data mining and machine learning research have developed powerful, automated, and scalable graph algorithms; however, these typically do not support interaction, nor are they designed specifically for sensemaking. Conversely, information visualization, which excels in interaction and developing user insight, often does not scale to the enormous information-rich networks we see today. In this dissertation, we combine visual analytics principles, modern graph visualizations, and new scalable graph exploration techniques.

Specifically, we have designed and developed a series of algorithms, novel approaches, and systems that support the investigation of large network datasets. Our work demonstrates new ideas that synergistically improve the speed and ease with which analysts can draw insight from network data.

1.1 Thesis Overview

Our work addresses three important research questions, summarized in Figure 1.1: (1) *How can we guide analysts in locally exploring million-edge graphs?* (2) after exploration, *How can we formulate and find graph patterns without writing complex querying code?* (3) *How can we summarize and explore subgraph matches?* We organize our research into three inter-related research thrusts, where each addresses one research question with: tools, methods, and systems (clicking a Thrust block below will jump to its text).

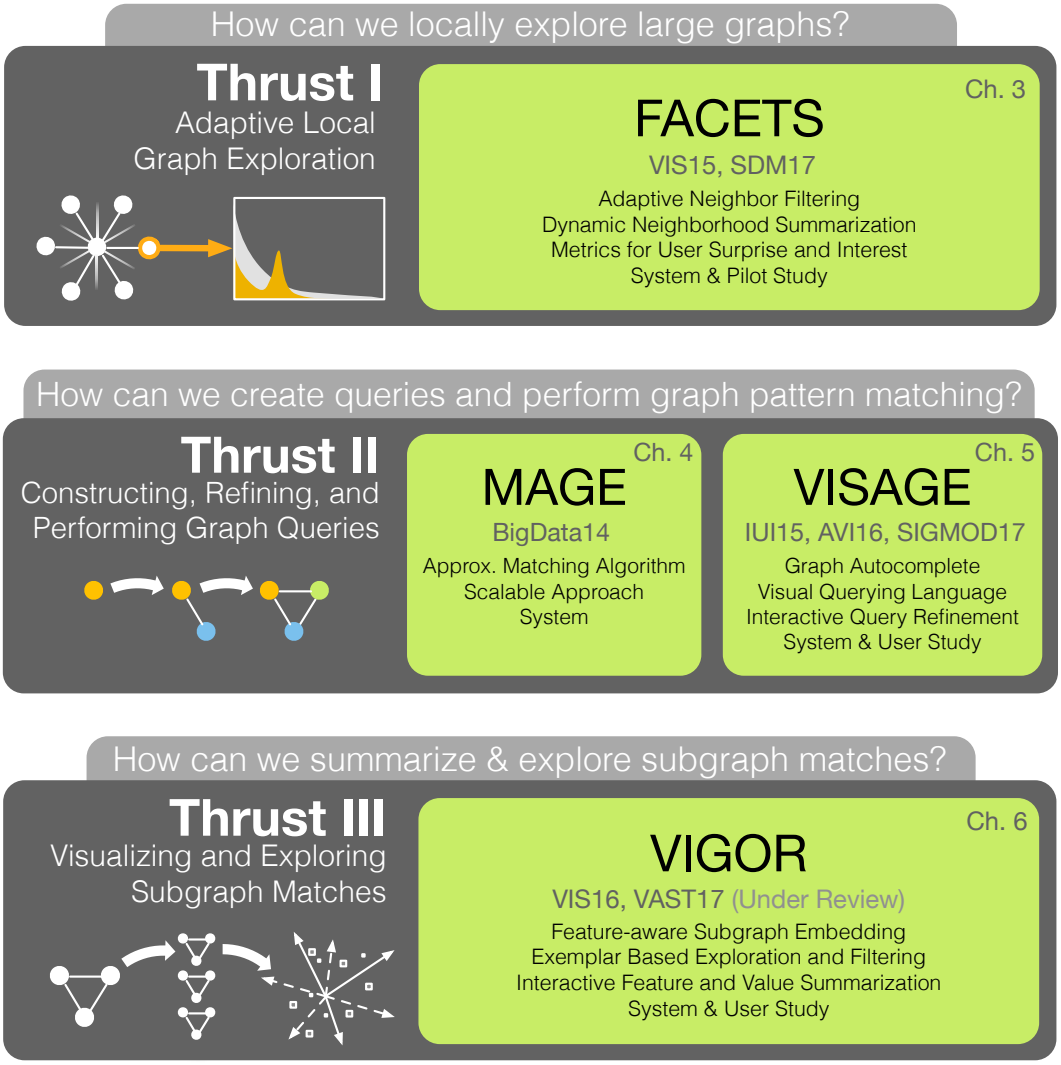


Figure 1.1: Our work is composed of three thrusts and four systems, motivated by important research questions shown in light gray above each thrust (clicking a thrust block below will jump to its place in the dissertation).

1.2 Thrust I: Adaptive Local Graph Exploration

Large graphs pose a significant challenge to visual analytics approaches, because the large number of nodes and edges in a graph often visually occlude each other (Figure 1.2). Even modest sized graphs are hard to utilize as a starting point for exploration. We propose to use adaptive local exploration to show only a subset of the whole graph and overcome the challenges of visual scale when facing massive graphs.

In **Thrust I**, we guide analysts towards nodes and node-neighborhoods with the FACETS system (pictured below in Figure 1.3 and covered in Chapter 3); enabling analysts to adaptively explore large million-node graphs from a local perspective. Its main features include:

- **Supplying local, dynamic summarization for node-neighborhoods.** Drawing an entire network often leads to incomprehensible hairballs (see Figure 1.2). By filtering out less relevant nodes from a neighborhood and summarizing the filtered content locally, analysts can quickly explore larger areas and extract important patterns.

- **Modeling user interest and surprise.** By only showing the parts of the graph that would be more interesting or surprising to the user, the visualization does not get too complex while still remaining interesting. Importance, surprise, and user-interest are all vital aspects of discovery, so we blend them into the results that are shown first to the user. FACETS uses Jensen-Shannon divergence over information-theoretically optimized histograms to calculate the subjective user interest and surprise scores.

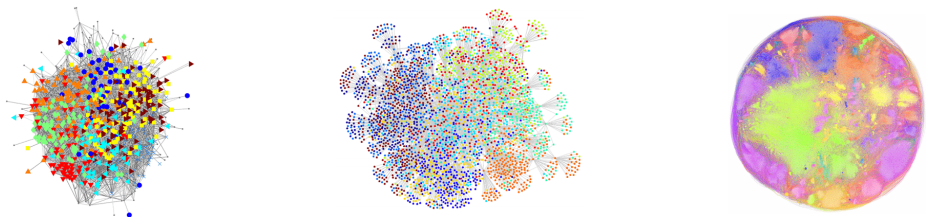
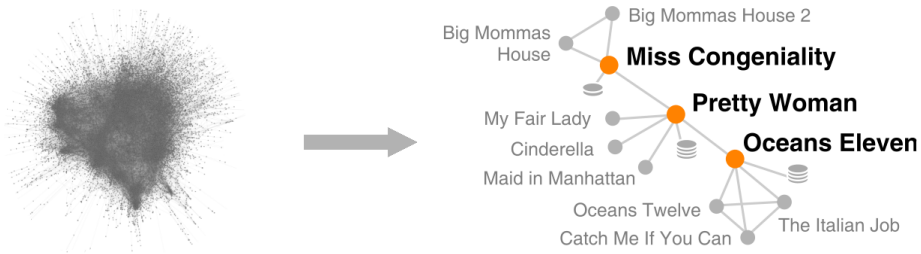


Figure 1.2: Visually overloaded global graph visualizations or *hairballs* from: the largest connected component from the Caltech Facebook social network [1]; a music-genre similarity graph [2]; and a visualization of the Steam social gaming network [3].



(a) *Complex* Global View **(b)** FACETS reveals insights

Figure 1.3: The Rotten Tomatoes movie similarity graph shown using conventional spring layout (an edge connects two movie nodes if users voted them as similar).(a) This global visualization does not provide much insight. (b) FACETS focuses on movies that are the most subjectively interesting, surprising, or both. In this example, FACETS suggests Pretty Woman (romantic-comedy) as an interesting, surprising related movie of Miss Congeniality (crime-comedy).

- **Actively adapting the exploration model to suit users' needs.** Analysts should be aware of emerging patterns in their foraging, which can subsequently be used to improve the filtering process on-the-fly. By being adaptive, it allows users to explore facets of the graph that are more subjectively interesting to them.

1.3 Thrust II: Interactive, Visual Graph Query Construction and Refinement

The foraging in Thrust I: Adaptive Local Graph Exploration, may yield interesting subgraph patterns. Other similar subgraphs may exist, but may be topologically far from the currently explored region. Graph querying extends the reach of local exploration once a pattern is known (or even partially known).

Constructing and refining graph queries requires complex querying languages [4, 5, 6, 7]; however, we show that queries can instead be formed via a visual, interactive system. Querying is often an iterative process that can benefit greatly from visual aids [8, 9]. Making such a system requires both algorithms and visualization working in tandem, as few algorithmic approaches have been designed for both interaction and visualization. This thrust focuses on algorithms to locate matches and techniques that help analysts construct and refine visual graph queries.

- **MAGE: Approximate graph matching (Chapter 4).** We present a scalable, ap-

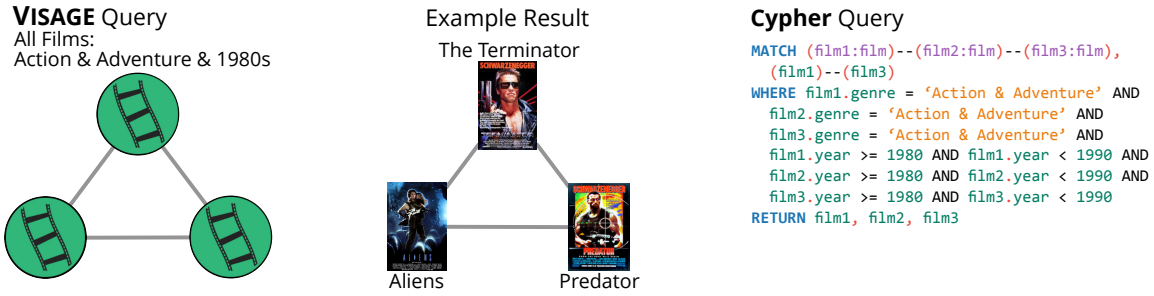


Figure 1.4: *Left*: a VISAGE query seeking three similar action films from the 1980's along with a result, found from the Rotten Tomatoes movie-similarity graph (an edge connects two movies if they are similar). *Right*: the equivalent query written in the Cypher querying language. VISAGE's interactive graph querying approach significantly simplifies the query writing process.

proximate subgraph matching approach that supports expressive queries over large, richly-attributed graphs. Approximate matching allows analysts to provide queries with missing or partial information and still receive results. This can improve the query refinement process, wherein an analyst has an inaccurate assumption in their query which is elucidated by their query results. We demonstrate MAGE s effectiveness and scalability via extensive experiments on large real and synthetic graphs, such as a Google+ social network with 460 million edges.

- **VISAGE: Interactive, visual query construction and refinement (Chapter 5).** Graph querying currently requires users to specify the structure of their query in a complex querying language. We show that human spatial reasoning can improve the query construction and refinement process when presented with a visual metaphor for their queries (Figure 1.4). By providing real-time data about an analyst's current query pattern as they construct it, our *graph-autocomplete* technique leads authors away from null-results and towards insight. We evaluated VISAGE with a twelve-participant, within-subject user study that demonstrates its ease of use and the ability to construct graph queries significantly faster than using a conventional query language. VISAGE works on real graphs with over 468K edges, achieving sub-second response times for common queries.

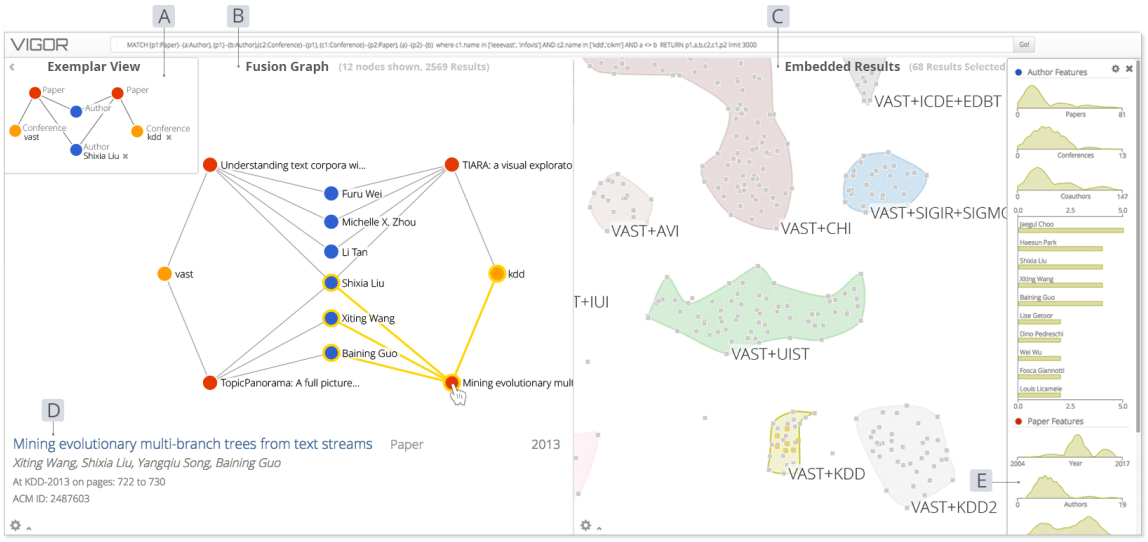


Figure 1.5: A screenshot of VIGOR showing an analyst exploring a DBLP co-authorship network, looking for researchers who have co-authored papers at the VAST and KDD conferences. (A) The Exemplar View visualizes the query, and (B) the Fusion Graph shows the induced graph formed by joining all query matches. Picking constant node values (e.g., Shixia) in the Exemplar View filters the Fusion Graph. (C) Hovering over a node shows its details. (D) The Subgraph Embedding embeds each match as a point in lower-dimensional space and clusters them to allow analysts to see patterns and outliers. (E) The Feature Explorer summarizes each cluster's feature distributions.

1.4 Thrust III: Visualizing and Exploring Subgraph Matches

After constructing a query an analyst will receive numerous subgraph matches (which we will call results). 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, various node features, and flexible structure across queries.

Our system, VIGOR (Figure 1.5), combines scalable algorithms and interactive visualization techniques to help summarize and compare large numbers of subgraph results (Chapter 6). Its key contributions include:

- **Visualization techniques for subgraph matches.** Currently, few graph querying systems offer more than tables, lists, or basic graph layouts for exploring results.

We introduce two new visualization techniques for analyzing results: (1) an induced result subgraph visualization that utilizes *soft-intersection* to highlight nodes that are commonly shared among result subgraphs; and (2) a top-down, high-level overview of all their results which enables them to handle complex grouping and comparison tasks.

- **An approach for exploring and filtering results through *exemplars* (familiar results).** Each subgraph result contains a multitude of nodes and edges. An analyst can interactively pick a known value (exemplar) for a node and immediately filter down the results (starting from all the results and filtering down by familiar values). An analyst to start with a whole result (in which all nodes have taken a value from the graph) and relax or abstract particular nodes, to see what similar results exist (starting from a known example and exploring the unknown).

- **A novel algorithm for feature-aware subgraph result embedding.** We introduce an algorithm to transform result subgraphs into continuous, high-dimensional vectors according to each result's node features (sometimes called properties) and structural features. This representation enables: (1) easier use with canonical machine learning and data mining techniques, (2) robust similarity and clustering, and (3) the ability to dimensionally reduce the results into 2D for summarization purposes.

1.5 How the Thrusts Interact and Benefit from Each Other: A Motivating Security Scenario

In this section, we present a scenario that demonstrates the synergy of our research thrusts. The scenario illustrates a fictitious security analyst, David, who is studying computer security threats on a company's intranet. His goal is to find infected machines and to understand the process by which they were compromised. The dataset consists of machines (nodes) and their networked communications (edges). Assume that the nodes have a plethora of information about the represented machine (the operating system, software versions, hardware specifications, administrator name, etc.). This scenario follows David as he utilizes systems, tools, and techniques from our three research thrusts (Figures 1.6 through 1.8 illustrate the scenario).

Several of David's coworkers, Alice, Greg, and Ziggy have noticed their machines are not working properly (Figure 1.6.1). With his domain knowledge, David quickly discovers several of the machines are compromised with malware.

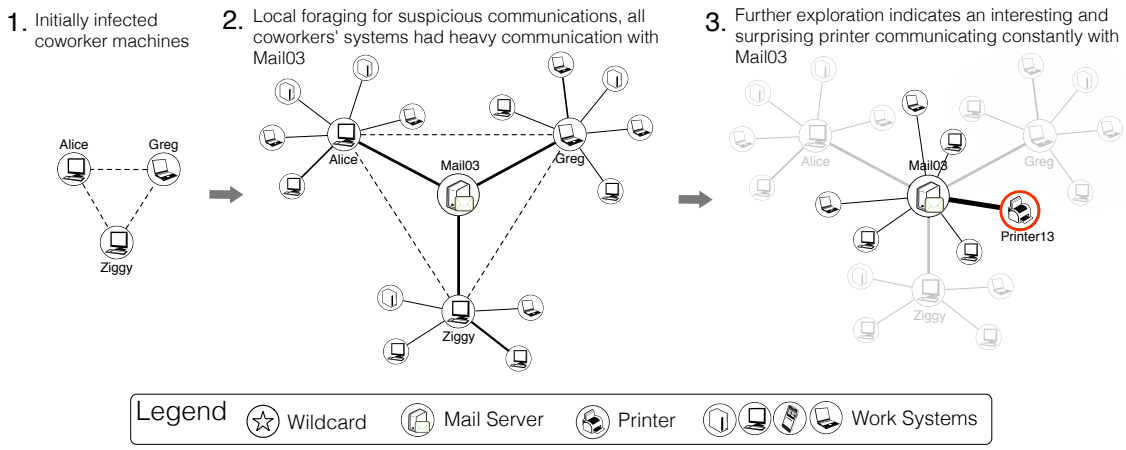


Figure 1.6: An Analyst, David, uses FACETS, a tool from Thrust I: Adaptive Local Graph Exploration, to explore a compromised computer network. (1) He explores compromised machines starting from a few know infected source machines (Alice, Greg, Ziggy). (2) David expands the neighbors of Alice, Greg, and Ziggy, FACETS shows only surprising and interesting neighbors. (3) He continues his search and discovers a suspicious printer, Printer13.

David uses our local exploration tool facets (Thrust I: Adaptive Local Graph Exploration) and begins by adding Alice, Greg, and Ziggy's machines (Figure 1.6.1). FACETS ranks the neighbors by interest (like Alice, Greg, and Ziggy's machines) and surprise (machines with unusual behavior compared to the global behavior).

He notices that Alice's, Greg's, and Ziggy's infected machines have recently communicated with the same mail server, Mail03 (in the center of Figure 1.6.2, the thickness of edges denotes the amount of communication).

He clicks Mail03 to show more of its most interesting and surprising neighbors (Figure 1.6.3). As David is clicking on nodes and exploring their details, FACETS is building an internal user profile for his exploration. He sees that a printer, Printer13, has been spamming Mail03 (red Printer-node in Figure 1.6.3). A printer! Our thin white duke is now alarmed; Printer13 was compromised and is using Mail03 to spread its control.

He now has some basic knowledge of a suspicious structure; this is where Thrust II (Interactive, Visual Graph Querying Construction and Refinement) works synergistically with his initial graph exploration from Thrust I (Adaptive Local Graph Exploration). He uses VISAGE (from Thrust II: Interactive, Visual Query Construction and Refinement) to

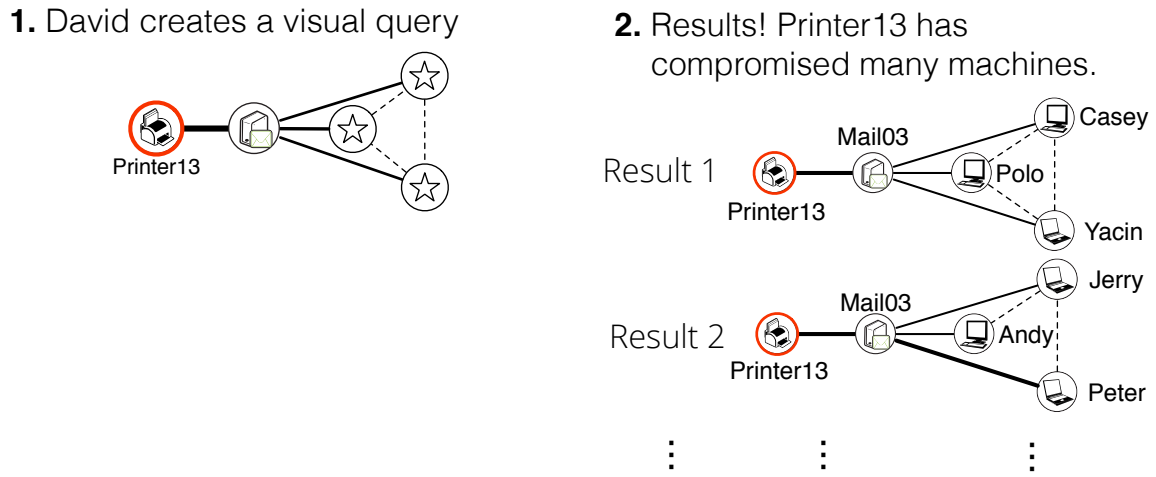


Figure 1.7: David wants to see if there are other infected devices connected to the mail server. (1) He constructs a query with VISAGE, but replaces Alice, Greg, and Ziggy's machine-nodes with wildcard nodes (nodes with a star), which could be any devices on the network. (2) VISAGE returns the results in a list of matches.

visually construct a query (without writing his query) starting specifically with Printer13 (as in Figure 1.7.1-1.7.2). To it he adds a node and selects its type as "Mail Server". To the mail server node he attaches a wildcard node that could be any machine (pictured as a node with a star in Figure 1.7.2). He copies this wildcard two more times and adds edges between them to form a triangle (dashed lines in Figure 1.7.2). He does this to match the original communication pattern among Greg, Alice, and Ziggy. He chose wildcards to find if other individuals are at risk. He has now created a query looking for any mail server that has been heavily communicating with Printer13 and with three other individual machines. He dispatches his query to our MAGE system, which finds the matches. MAGE (from Thrust II) is an algorithm which finds exact and approximate subgraph matches in large networks.

The results flood into view (listed in Figure 1.7.3) via a conventional list-view provided by VISAGE. The list view shows each result separately, allowing him to see which machines fit his infection-pattern. Given only the table and list visualizations, its a challenge for him to determine what groupings of similar results occur or how a particular node value appears among the results. He wonders if other systems besides the printer are involved. He modifies his VISAGE query by replacing Printer13 with a wildcard (Figure 1.8.1-1.8.2) and dispatches it back to MAGE.

With the more abstracted query, there are significantly more matches. The problem looks serious; he does not have time to go through each result separately. David then uses VIGOR from Thrust III: Visualizing and Exploring Subgraphs to switch from the list view to a summary view (Figure 1.8.3), where each result is plotted as a single point, providing a high-level overview of result similarity. The clusters of 2D points, represent sets of similar results from a dimensionally-reduced graph embedding (described in Chapter 6). By selecting different clusters of nodes he can quickly evaluate other potential abstractions of his initial intuition about the compromised systems. Figure 1.8.3 shows some of the other clusters that he has found. The compromised systems are all over the company, not

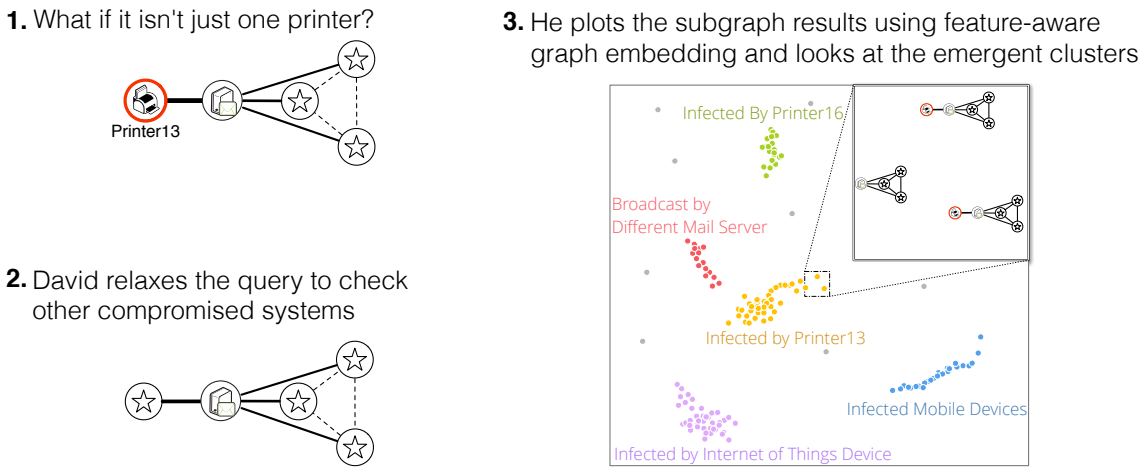


Figure 1.8: Analyst David abstracts Printer13 in his query (1) to a wildcard looking for any system with similar communication patterns (2). He receives the results of his abstracted query and views them in a feature-aware subgraph result embedding (3). Each result subgraph is reduced to a point so that similar results appear close to one-another. He uses this view to find other infected systems.

just within a single department or group. He notifies management.

Exemplar-based filtering allows David to select known values (like Printer13) in his query and filter down to only results that have the exemplar node (i.e., only mail servers and systems compromised by Printer13). This makes classifying where the damage has happened much faster and easier than a conventional table.

Using the tools developed in our three research Thrusts, David was able to detect an initial pattern, formulate queries around it, investigate other similar communication patterns, and quickly discern which groups have compromised systems. He has used graph sensemaking to help detect a serious threat by weaving our three research thrusts.

1.6 Thesis Statement

We blend methods, techniques, and principles from *visual analytics* and *data mining* to create new interactive, scalable tools that enable analysts to explore and analyze large networks more easily, through:

1. a novel adaptive local network exploration paradigm and new measures of surprise and interest to guide user towards interesting content;

2. a synergistic combination of algorithmic and interactive visual techniques to empower analysts to construct and refine expressive queries without writing complex querying code; and

3. new result summarization techniques based on feature-aware subgraph-result embedding and clustering that allows analysts to quickly compare numerous graph query results.

1.7 Research Contributions and Impact

The goal of our thesis is to combine the strengths of data mining, visual analytics, and graph databases to improve graph sensemaking through network exploration and querying. We contribute to several facets of data mining, and visual analytics by combining them together.

For Data Mining:

- **Algorithms.** We introduce FACETS (Chapter 3) which utilizes fast, dynamic rankings of node neighborhoods to lead analysts towards interesting content. We design and develop a highly scalable algorithm in MAGE (Chapter 4) for approximate subgraph matching on large networks with node and edge attributes, wildcards, and multi-attributes. MAGE uses a novel approach based on the linegraph transformation to embed edge attributes. In VISAGE (Chapter 5), we create algorithms to support dynamic interactive graph querying in real time by transforming visual queries into written queries in two querying languages. In our VIGOR system (Chapter 6), we introduce a novel, feature-aware subgraph result embedding. This embedding uses semantic node-feature data as well as topological data when creating a high-dimensional result embedding.

- **Systems.** In this thesis work we have developed four systems. We contribute them to the research community as planned open-source projects: FACETS for adaptive local exploration (Chapter 3), MAGE for approximate subgraph matching (Chapter 4), VISAGE interactive visual graph query construction and refinement (Chapter 5), and VIGOR exploratory visualization of subgraph results (Chapter 6). VISAGE won the SIGMOD'17 Best Demo Honorable Mention award.

- **New Metrics to Determine User Interest.** In FACETS, Chapter 3, we contribute two new metrics based off of the Jensen-Shannon divergence of feature-neighborhoods:

(1) for subjective user interest modeling and (2) for measuring a nodes rarity and surprisingness against a network.

- **Novel subgraph-result embedding** We introduce an algorithm to group results by node-feature and structural result similarity and embed each result in a low dimensional representation. By grouping similar results into clusters and making cluster comparisons easy, analysts can quickly detect and understand underlying patterns across their results.

For Visual Analytics:

- **New Adaptive Graph Exploration Paradigm.** We contributed a survey that summarizes and discuss many challenges and opportunities for new network analytical systems (Chapter 2). FACETS is a new step towards an adaptive-exploration paradigm in visual analytics, exploiting the data generated during interactions between a software system and a user to aid the exploration of data.

- **New Interactive Graph Query Construction and Refinement via Graph Autocomplete.** VISAGE (Chapter 5) represents a major step towards interactive, visual graph querying. It provides a new, highly-usable querying paradigm, that uses simple drag-and-drop interaction to construct complex queries, rather than having to learn a querying language VISAGE introduces a novel interaction technique called *graph-autocomplete*, which guides analysts away from over-specifying their queries. VISAGE outperforms conventional querying writing and refinement in a laboratory study for both expert and non-expert subjects.

- **Visual Result Exploration** VIGOR introduces a new overview mechanism to visualize large numbers of subgraph results (Chapter 6). It also employs an exemplar filter to explore the results by filter down to only results with familiar nodes.

- **Scalable Interactive Tools.** Our interactive systems advance the state of the art, by facilitating real-time discovery and exploration in graphs with millions of edges (e.g., VISAGE and FACETS). Empirical runtime analyses demonstrated FACETS s practical scalability on large real-world graphs with up to 5 million edges, returning results in fewer than 1.5 seconds. We demonstrated realtime querying with VISAGE, with sub-second querying times on a real Rotten Tomatoes movie graph with over 170 thousand relationships.

We believe that our thesis will catalyze and accelerate new research and innovation bridging the areas of graph data mining and visual analytics. We hope that this human-in-the-loop data mining work inspires researchers and practitioners to consider the rich potential of working in the intersection of data mining and visual analytics.

Our research allows analysts to forage for graph insight faster and, once interesting patterns have been uncovered, to expand on them with interactive, visual graph querying. All without the need to learn a new querying language.

CHAPTER 2
BACKGROUND & PRIOR WORK

In this chapter we will cover the related work necessary for our research. Because our research combines visual analytics, data mining, databases, and sensemaking, we will cover a large variety of research areas.

2.1 Scalable Graph Exploration and Visualization: Sensemaking Challenges and Opportunities

In this dissertation we focus on local graph exploration techniques. Local views are visualizations in which only a subset of the entire graph data is shown. Because only a portion of a much larger graph is displayed, these approaches tend to improve visual comprehension, require less computation, but show only a subset of the total information. Many local algorithms need only to run on subgraphs, potentially improving scalability.

2.2 Graph Visualizations & Global Exploration

Both Herman [11] and Landesberger [12] have surveyed the rich variety of static graph visualizations. Beck et al. followed up on this surveys and investigated visualization techniques for dynamic graphs [13].

Global visualizations are often very challenging due to the amount of space needed to layout all the nodes and edges [14, 15, 16]. Many graph datasets do not contain spatial node positions, leaving their spatial layout as an exercise for the analyst. Significant research has been done by graph drawing communities investigating how to lay out and summarize entire graphs [17, 14, 15, 16].

Chapter adapted from work at BigComp'15 [Paper Link] [10]

For graphs that are too large to display at once in full detail, several classes of approaches have been proposed to make them more manageable for exploration: sampling, filtering, partitioning, and clustering. We use many of the ideas in our works VISAGE, VIGOR, and FACETS which require visualizing and interacting with subgraphs. In the following sections we will elaborate on which areas we have drawn inspiration in our own works.

2.3 Free Exploration & Targeted Discovery

Local exploration was first investigated on hierarchical graphs in [18] and later expanded on by [19] to incorporate the idea of "degree of interest" to help users identify which nodes to explore. Systems like Apolo [20] do not impose an hierarchy on the data, allowing users to freely define their own clusters, which Apolo incorporates into its machine learning algorithms to infer which nodes the users may want to explore next. We draw directly on these ideas in FACETS to show users a mix of potentially interesting and surprising nodes based off their current exploration.

Information retrieval research has focused on scalable approaches to analyze the web-browsing and paths users traversed as they explored the web for millions of users. Click trails have been used to improve the ranking of search results [21, 22]. In many cases, the destinations of such trails can be used directly as search results [23], or even to teleport the user directly to the desired page [24]. FACETS attempts to dynamically infer these patterns by analyzing the ongoing trail formed by the analysts current graph exploration. It uses emergent patters in the trail to show only a subset of a node's neighbors. West et al. studied users' abilities and wayfinding techniques as users crawled Wikipedia [25]. They observed a trade-off wherein users would prefer conceptually simple solutions at the cost of efficiency.

2.4 Graph Sampling & Filtering

Instead of drawing the entire graph, many approaches sample or filter (stochastically or deterministically) a subset to reduce a graph's size. There are numerous techniques which aim to maintain different structural properties, while randomly sampling from a graph [26, 27, 28, 29]; [14] and [30] compare them on real graphs. Many recent techniques are surveyed in [31]. Lee et al. investigated statistical properties of sampling techniques and characterizes techniques on how they bias topological properties like betweeness centrality, assortativity, and clustering coefficient [32].

We use graph sampling ideas in VISAGE to create graph autocomplete, a method that samples graph results to help users avoid over-specification and null-results. In our ongoing multi-result visualization research, we have experimented with sampling techniques to offer the user some context from the original graph along with their results.

Jia et al. have shown that filtering by the approximate betweeness centrality will reduce the graph size while still maintaining the essential structure of the graph [33]. Filtering can even be employed to improve the performance of graph-based machine learning models [34]. This helps reduce visual clutter from large numbers of edges, which *edge bundling* [35] may be able to simplify. Edge bundling bundles groups of overlapping edges together to reduce the edge-clutter and clarify the underlying structure. There are numerous edge bundling variants: force-directed [36], multilevel agglomerative [37], hierarchical [35], and skeleton [38].

Our FACETS system uses adaptive filtering to show the user the most interesting nodes based on their current exploration. Instead of filtering by centrality our filter blends interesting, surprising and important nodes to help the analyst make sense of their data.

2.5 Partitioning & Clustering Graphs

2.5.1 By Structure

A common approach to creating an overview graph is to use partitioning methods on the graph and visualize the partitions [39, 40]. Often particular graph topologies can be leveraged to improve the partitioning; as with bipartite graphs in [41] or meshes and irregular graphs in [42]; however, partitioning scale-free networks often results in exceedingly poor partitions [43]. PULP [44] was designed to partition small-world networks. Partitioning can be useful when organizing large graphs, and may be needed in our upcoming work on visualizing graph uncertainty.

2.5.2 By Attributes

Another approach is to create clusters of nodes with similar attributes or to use online analytical processing (OLAP) techniques to roll-up all nodes with a common attribute.

In [45], Tian et al. demonstrate *SNAP*; which creates a summary graph by allowing user-specified attributes to determine node-node similarity; and *k-SNAP* which automatically generated subgroups allowing a user to drill-down or roll-up levels of summarization. Neville et al. used relational attribute information to create clusters from a the rich relational data for text and link mining [46]. We have started experimenting with attribute-centric clustering techniques for our ongoing work visualizing multiple results.

2.5.3 Using Both Structure and Attributes

Combining both structural and attribute information yields a reduced version of the graph where the clusters are both structurally tight and of similar attributes [47, 48]. Zhou et al. proposes a novel distance measure that combines both structural distance as well as node attribute similarity [48]. PivotGraph [47] aggregates nodes and edges based on their attributes; however, it uses a grid-based layout to focus on the relationship between nodes'

attributes and connections.

Clusters may also be human-generated, as in [20, 49]. Allowing users to generate and customize their own clusters makes exploration more flexible to changes in input datasets. Rather than relying on a force-directed layout, Schneiderman and Aris propose a static graph layout called *semantic substrates* [49] which are user-defined, non-overlapping regions in which the nodes are placed according to their attributes.

This area is important to our proposed work in Chapter 6. Both visualizing approximate subgraph matches and visualizing large numbers of subgraphs can benefit greatly from clustering techniques.

2.6 Graph Models, Storage, & Databases

One considerable challenge when querying graphs is storing the graph data. Storing graph data often follows one of two routes:

1. **Tuple-based storage** - Each relationship of the network is captured as a tuple between two entities, or an entity with itself. The most common is *Resource Description Framework* (RDF), but other tuple-based models predated RDF like the Logical and Functional Data Models respectively [50, 51].

2. **Explicit graph data models** -Relationships are stored explicitly as adjanceny matrices, edge-lists or other similar explicit topology-capturing data structures. Models like GRAS [52], G-BASE [53], Gram [54], and GraphDB [55] represent the foundation of explicit graph data that many querying systems use today. Many of these approaches fall into the category of "not only SQL" or "Non SQL" *NoSQL* databases many of which are covered here [56].

While our work in this dissertation does not explicitly focus on the development of novel graphical models, we build on the expertise and research of others in all of our publications. For our works FACETS and MAGE works we leverage explicit graph models through the

compressed sparse row format heavily [57]. Our VISAGE, CHOPSHOP, and VIGOR approaches work regardless of the graph database system by supporting modules for multiple graph databases. They will be discussed in greater detail in Chapters 5 and 6.

2.6.1 Querying Languages

Query By Example [58], 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 [59], which uses visual glyphs to create visual database queries. Both [60] and [61] avoid complex data schemas in favor of graphical widgets. For a further detail of visual querying languages on relational databases see [62]. Data storage techniques like the extensible markup language (XML) and resource description framework (RDF) have spurred other querying languages like XQUERY [4], XPATH [5], SPARQL [6]. Both [63] and [64] propose graphical querying languages for XML, while Hogenboom et. al propose one for RDF data [65]. Our work builds on visual querying by using visual metaphors for both constructing graph queries and displaying the results.

2.7 Subgraph Mining & Graph Querying

Domains from bioinformatics to intelligence analysis often seek particular subgraphs from their data. Graph pattern matching is a variation of the subgraph isomorphism problem, an NP-complete task of determining if a given graph is a subgraph of another graph [66]. Exact graph pattern matching is computationally expensive and hard to parallelize. There are numerous techniques to extract all the matching subgraphs from a network; however they often follow one of the following paths:

1. **Structural Matching** - Many domains rely on exact structural matching. Ullmann first published an algorithmic solution to tackle this combinatorial problem [67]. Mesmer and Bunke later showed that Ullmann's algorithm has exponential worst-

case time complexity and introduced a quadractic time algorithm [68]. Their solution operates in polynomial time; however, it requires an exponentially-sized decision-tree, making it unrealistic for large graphs. New approaches have extended these results: singular value based matching [69], weighted graph matching with eigende-composition [70], inexact mathcing with genetic search [71], and many others some of which are surveyed in [72].

2. **Semantic Matching** - Many real world networks are rich with types and features on both vertices and edges. One sensible approach is to search for approximate matches while another is to leverage attributed subgraphs to speed up search. The TRAKS system, [73], allows a user to query to match templates using an ontology to detect security risks. TMODS, [74], uses genetic algorithms to find pattern matches in attributed graphs.

3. **Hybrid Structural-Semantic Matching** There are a few recent systems that offer approximate subgraph matching, which all focus on large scale techniques for semantic and structural matching. These include G-RAY [75], MAGE [76], Graphite [77], NeMa [78], TALE [79], and TopKDiv [80]; to name a few. This is essential in scenarios where the user already knows of an interesting pattern exactly or approximately, and wants to find where or how often it occurs in a larger graph. Many of these systems do not focus on the visualization of the query and results, but rather on the algorithmic and data mining challenges. Fan et al. exploited this idea in order to hide the large latencies of graph querying by constructing partial results as a user specified their query pattern [81, 80].

2.7.1 Visual Graph Querying

Several recent systems focus on providing user interfaces for graph query construction. GRAPHITE [77], allows users to visually construct a graph query over categorically at-

tributed graphs. VOGUE [82], 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 [83]. These tools formed the foundation that lead us to create our VISAGE (Chapter 5) and VIGOR (Chapter 6) systems.

2.8 Frequent Subgraph Mining & Network Motifs

Many works have focused on the discovery of common subgraphs from within much larger graph datasets, which could help discover abnormal activities in the networks, e.g., auction fraud [84], insider trading [85], detecting patterns in protein networks [86], or insider threats in a company [87]. This explorative process requires almost no foreknowledge of the input graph. Originally coined in [88, 89], *network motifs* are common subgraphs or patterns that occur "unusually" often in a network when compared against random networks of the same size.

Generating the network motifs is a computationally expensive procedure involving subgraph enumeration and aspects of graph similarity from subgraph isomorphism, current approaches can detect motifs with dozens of nodes, for modestly sized graphs [90]. Milo et al. use a scanning approach for all n-node subgraphs and then compares the occurrence of such n-node graphs with their chance to occur in a random-network [89]. Yan et al. created *gSpan*, short for graph-based substructure pattern mining, which discovers frequent graph substructures without the need for a prebuilt candidate list [91]. Grochow et al. further improved motif detection scalability in [90], by using subgraph enumeration and symmetry breaking which intelligently eliminates repeated comparisons, yielding significant speedups.

Related to motif identification, recent research [92] proposed to develop a vocabulary of common subgraph patterns that can be used to summarize the global graph. Dunne and Shneiderman [93] 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 MAVisto [86].

These techniques provide a collection of data mining and visualization techniques to show subgraph matches embedded in the graph from which they were mined and provide statiscal measures for motif or pattern significance. This is directly applicable to our result visualization work in Chapter 6. We want to provide users with structural context from their original graph given a set of subgraphs and motif mining may offer a principled way to pick the right neighbors to show.

2.9 Statistical Relational Models and Querying

Statistical relational learning formulates domain models that exhibit both uncertainty and complex, relational structure. Statistical relational models (SRMs) have been applied to many graph and relational data models [94, 95, 96]. We leverage SRM techniques in our CHOPSHOP work to predict how likely certain constraints and graph structures are. We use these likelihoods to help a user get more results from their query, by offering relaxations and modifications to constraints that the user has performed. The *Why Not?* system [97] attempts to solve a similar problem, to help users whose query has returned little or nothing. Their approach uses the query evaluation plan in a SQL relational database to locate over-specified constraints and subsequently allows the user to remove them.

Probabilistic database systems use a similar data-driven representation Utilizing the underlying distributions was used in [98] to create explanations for SQL queries. The explanations are tuples which critically effect the output of complex queries (i.e., critical aspects of the results that pose an *explanation* for the observed behavior).

2.10 Graph Kernels and Embedding

Graph kernels are a class of function that computes the inner product on entire graphs and are often used to calculate the similarity between a pair of graphs. There are numerous

types of graph kernels: Weisfeiler subtree [99], path-based [100], graph edit distance based [101], marginalized [102, 103], and many others [104]. We use graph kernels to create the input pairwise distances for dimensionality reduction (covered in the next sub-section) used to plot subgraph results in VIGOR.

Both [105] and [104] use the structure to create the embedding while NetSimile, [106], uses extracted features. Grover et al. released a technique called **node2vec**, a graph embedding that smoothly blends local and global characteristics when creating a continuous, high dimensional representation [107]. A similar approach, **subgraph2vec** was also suggested, which learns latent representations of subgraphs in a continuous, high dimensional space for use in deep-learning applications [108]. Van den Elzen et al. used graph embedding to plot the changes in dynamic graph snapshots over time [109]. We draw on some of these ideas in VIGOR to collapse each subgraph result down to a point, the details of which will be discussed in Chapter 6.

2.11 Dimensionality Reduction

Dimensionality reduction takes a collection of points and projects them into a low dimensional subspace, so that characteristics of the high dimensional points are maintained. There are numerous linear and non-linear dimensionality reduction techniques: *principal component analysis* (PCA) [110, 111], *kernel-PCA* (KPCA) [112, 113], *multidimensional scaling* (MDS) [114, 115], *t-Distributed Stochastic Neighbor Embedding* (t-SNE) [116], *local linear embedding* (LLE) [117].

Many of the approaches run in quadratic time with respect to the number of data points, which becomes prohibitively expensive. Variants of these approaches reduce the computational demands like *Randomized* PCA [118, 119], MDS [120], *Barnes-Hut*-SNE [121], and LLE [122].

Approaches like MDS and t-SNE have an additional advantage, they allow non-Euclidean distance calculations when determining the *distance* between points. Other

distance metrics can provide better performance: Canberra [123], Kolmogorov-Smirnov [124], or the *degree distribution quantification and comparison* measure [125]. Our system VIGOR (Chapter 6) makes use of dimensionality reduction to represent each subgraph result as a point in 2D. Because dimensionality reduction techniques apply differently to various scenarios, we make multiple options available to the analyst during runtime.

2.12 Graph Interaction Techniques

The HCI and visualization communities have researched many useful interaction techniques [126, 127, 128, 127]. Recently there has been significant focus on network interactio and exploration [128, 129, 130] Lee et al. taxonomized common graph visualization interactions in [131]. They separate low level tasks into topological, attribute, and browsing based groups. This research is important to our thesis, because usable, intuitive interfaces can greatly streamline the process of exploring data.

2.12.1 Common Interaction Techniques

User interaction in graph visualizations is essential in all graph exploration tasks. Canonical graph interaction techniques such as brushing, linking, panning, and zooming appear consistently in graph visualizations [132, 133, 134, 135, 136]. We offer variations of these in both FACETS and VISAGE, and we will make further use of them in our proposed and ongoing work.

Interaction is often used to change the visual representation of network data from a network diagram to a matrix or visa-versa as in Node-Trix [137] and with elastic hierarchies [138].

2.12.2 Lenses & Selections

Another common approach is to provide "details on demand" through a simulated lens that provides a detailed view when placed over dense areas. Lenses have proven effective for

numerous graph applications [18, 139, 19, 140, 141, 130]. Lenses often highlight geometric aspects of the data, but they may also highlight other elements, like *Color Lenses* [136] or *Excentric Labels* [142, 143] which provide greater detail in data-dense visualizations like matrices and tables.

There are two main methods for node and edge selection: pointer-selection and query-selection. In pointer-selection, the pointer is used to: manually click to select, drag a selection, draw a selection lasso or brush a selection. Query-selection usually uses a query language or filtering interface to let the user specify which nodes they want selected based on node or edge level attributes. Query-selection can be especially useful in cases with rich multivariate node and edge data.

2.12.3 Structural & Topological Navigation

Topological navigation uses the graphs structure to show and hide portions of the graph based on the connections between nodes. Often this is used so that only a local area of interest is displayed. This *neighborhood traversal* technique can be very effective means to explore a graph using local topological jumps [144].

TreePlus [129] uses a tree structure to aid in users' exploration of hierarchically clustered graph data. By letting users selectively *grow* the hierarchy, TreePlus strikes a balance between detail and intuition by offering excellent readability, layout stability, and the users' perceptions of tree structure.

In the case of networks with scale-free or near scale-free degree distributions (and other graphs with high degree nodes), pure topological browsing is insufficient, because drawing a single node's neighbors may be drawing a large portion of the graph. Thrust 1 aims to overcome this issue with adaptive filtering and node-neighborhood summaries. We demonstrate this in our FACETS system.

2.12.4 Degree of Interest Navigation

Degree of interest (DoI) techniques use a function to hide parts of the graph that are uninteresting to the user [18, 145, 19, 146]. A DoI function evaluates the importance of nodes based on an initial node or group of nodes and produces a ranking for related nodes. Neighborhood traversal can be expressed as a simple DoI function. While the DoI functions proposed originally in [18] used a form of graph distance, other graph-attributes can be used to capture user interest [19].

DoI functions can be dynamic and tuneable: Abello et al. created a modular DoI for large dynamic networks; wherein they provide the user an interactively defined DoI to improve a user's ability to track critical dynamic elements of their network [147]. The Apolo system integrates machine learning to infer multiple types of DoIs simultaneously [20]. Recently, *Entourage*, a tool for visualizing biological pathways, uses contextual information provided by the user to visualize interdependencies among pathways [148].

Our first thrust for adaptive local exploration relies heavily on the previous research into DoI navigation. FACETS uses a dynamic DoI that combines both interest and surprise to show only a subset of nodes. Our dynamic DoI is updates based on the user's interactions with displayed content in the system.

Thrust I

Adaptive Local Graph Exploration and Navigation

OVERVIEW

Given the enormity of modern network datasets, there are many challenges involved in supporting graph sensemaking. Everyone has different prior knowledge, goals, and approaches to solving a problem. These facts drove us towards research to: flexibly create data landscapes that are adaptively tailored to analysts needs as they explore, and rank nodes by more than canonical importance or centrality measures.

Conventionally, the mantra "overview first, zoom & filter, details-on-demand" in information visualization relies on peoples perceptual ability to understand a global or summarized version of their dataset. Because the large number of nodes and edges in a graph often occlude each other, even modest sized graphs are hard to utilize as a starting point for top-down exploration.

In the following **Thrust I**, we present FACETS (Chapter 3), a system which enables analysts adaptively explore large million-node graphs from a local perspective. We overcome many challenges of the *top-down exploration paradigm* by starting from a single node and showing only a subset of interesting and surprising neighboring nodes at each step of the way. We contribute novel ideas to measure user interest in terms of how surprising a neighborhood is given the background distribution, as well as how closely it matches what the user has chosen to explore.

CHAPTER 3

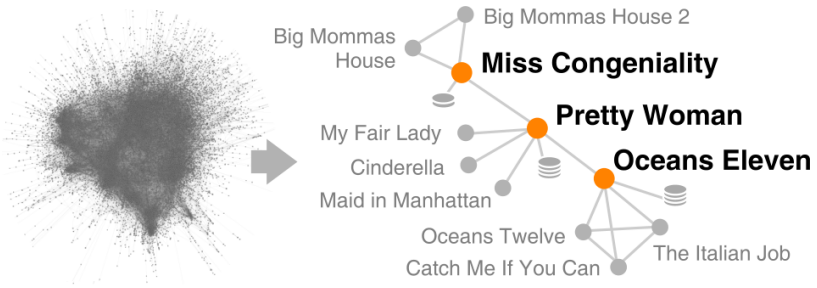
FACETS: ADAPTIVE LOCAL EXPLORATION OF LARGE GRAPHS

Large graphs are ubiquitous. They are natural representations for many domains, and hence we find graph structured data everywhere. As data collection becomes increasingly simple, and many domains remain complex, real-world graphs are rapidly increasing in size and data richness. These graphs may have thousands or more of attributes and have over millions and billions of nodes and edges. It is fair to say that many graphs are in fact *too big*; exploring such large graphs, where the goal of the user is to gain understanding, is a highly non-trivial task.

Visualization is perhaps the most natural approach to exploratory data analysis. Under the right visualization, finding patterns, deciding what is interesting, what is not, and what to investigate next should become easy tasks — in a sense the answers "jump to us" as our brains are highly specialized for analyzing complex visual data. It is therefore no surprise that Shneiderman's mantra of "overview, zoom & filter, details-on-demand" [151] has proven to be successful in many domains [152, 153, 151].

Visualizing large graphs in an intuitive and informative manner has proven to be difficult. Even with advanced layout techniques (e.g., those covered in [33, 15]), plotting a graph can create a hard-to-read cluster of overlapping nodes and edges, from which little can be deduced [152, 154]. This is the case even for graphs with only thousands of nodes (see Figure 3.1(a) for an example). Instead of plotting the whole graph, visualizing only part of the graph seems more promising [19, 20, 155]. However, as many real world graphs are scale-free (follow a power law degree distribution [156]), selecting relevant subgraphs to visualize can be challenging [20], since in such graphs, a single-hop neighborhood ex-

Chapter adapted from work at SDM'17 [Paper Link] [149]
Poster Version of the Work at IEEE VIS'15 [Paper Link] [150]



(a) *Complex* Global View **(b)** FACETS reveals insights

Figure 3.1: (a) The Rotten Tomatoes movie graph shown using conventional spring layout (an edge connects two movie nodes if some users voted them as similar). Even for this relatively small graph of 17k nodes and 72k edges, a global visualization does not provide much insight. (b) A better way, using our FACETS approach, focuses on movies that are the most subjectively interesting, surprising, or both. For example, FACETS suggests *Pretty Woman* (romantic-comedy) as a interesting, surprising related movie of *Miss Congeniality* (crime-comedy).

pansion from a node can be visually overwhelming.

We take a different approach. We propose to *adaptively* explore large graphs from a *local* perspective. That is, starting from an initially selected node — e.g., explicitly queried by the user, or proposed by an outlier detection algorithm [157] — we only show the most interesting neighbors as the user explores the graph from node to node. We identify these by their *subjective interestingness* based on how surprising their and their neighbors' data distributions are (e.g., do neighbors' degree distributions follow a power law like when considering all nodes?), as well as by how similar those distributions are compared to those of the nodes the user has explored so far. By only showing those parts of the graph that are most interesting to the user, we keep the view clean. By being adaptive, FACETS allows users to explore facets of the graph that are most interesting to them.

We call our adaptive approach FACETS — our idea is a significant addition to existing works that aim to recommend individual nodes to users (e.g., with centrality measures [147, 20, 19]); instead, we steer users towards local regions of the graphs that match best with their current browsing interests, helping them better understand and visualize the graph at the same time. We focus on ranking nodes based off how interesting an unexpected their neighborhoods are.

3.1 Illustrative Scenario

To illustrate how FACETS works in practice, consider our user Susan who is looking for interesting movies to watch (see Figure 3.1), by exploring a Rotten Tomatoes movie similarity graph with 17k movies. In this graph, an edge connects two movie nodes if users of Rotten Tomatoes voted them as similar films. Susan has watched *Miss Congeniality*[1], a crime-comedy that stars Sandra Bullock as an FBI agent who thwarts terrorist efforts by going undercover, turning her rude unflattering self into a glamorous beauty queen (see Figure 3.1b). FACETS simultaneously suggests a few movies that are interesting and surprising to Susan.

Matching Susan's interest, FACETS suggests the *Big Mommas House* series, which also have undercover plots and are interestingly like *Miss Congeniality*. They both share low critics scores, but high audience scores (i.e., most critics do not like them, but people love them). To Susan's surprise, FACETS suggests *Pretty Woman*, which is quite different (thus surprising) — a romantic-comedy that has both scores from the critics and the audience. But, there is still more subtle similarity (thus still drawing Susan's interest); both films share a Cinderalla-like storyline, which explains why the two movies are connected in the graph: Sandra Bullock goes from a rude agent to a beauty queen; in *Pretty Woman*, Julia Roberts goes from a prostitute to a fair lady. In fact, *Pretty Woman* is a classic, exemplar romantic-comedy; many movies follow similar story lines (e.g., *Maid in Manhattan*). Thus, *Pretty Woman* has very high degree in the graph, unlike *Miss Congeniality* which is a niche genre; this also contributes to *Pretty Woman*'s surprisingness.

Through *Pretty Woman*, FACETS again pleasantly surprises Susan with *Oceans Eleven*, which also stars Julia Roberts, and is in a rather different light-hearted crime or heist genre, introducing Susan to other very similar movies like *Oceans Twelve* and *The Italian Job*. Figure 3.1(b) summarizes Susan's exploration. If Susan were to use a conventional visualization tool to perform the same kind of movie exploration, she would likely be completely

[1] One of the most frequently rated movies on Netflix

overwhelmed with an incomprehensible graph visualization (as in Figure 3.1(a)).

3.2 FACETS's Contributions

The key contributions of this chapter include:

- A framework for locally exploring a graph without clutter, showing only the most subjectively interesting nodes, and hence being *adaptive* to the users' interests.
- A formal notion of subjective interestingness for graph exploration taking both divergence between local and global distributions, and similarity to explored nodes into account.
- A measure of surprise over neighborhoods — rather than local node attributes — to draw users in the direction of graph areas with unexpected content.
- A highly scalable method, FACETS, for adaptively exploring very large graphs in a visual environment. Experimental evidence demonstrates the effectiveness of FACETS.

The rest of the chapter is organized as follows: First, we discuss related work and then, in the *Model* section, we formalize the problem, introduce our notions of interestingness, and propose our FACETS solution. In the subsequent *Approach* section we present our ideas as an integrated approach, with visualization and algorithms working closely together. In the *Experiments* section, we evaluate FACETS on large real-world datasets through multiple complementary ways, including a small observational study, run time and scalability analysis, and three case studies. Finally, we conclude.

In this section, we formalize the problem we aim to solve through our FACETS approach to achieve adaptive exploration. Then, we describe our main approach, and proposed solutions. To enhance readability, we have listed the symbols used in this chapter in Table 3.1. The reader may want to return to this table for technical terms meanings used in various contexts of discussion.

Table 3.1: Symbols & Notation

Symbol	Description
v_i	Node i
D_{JS}	Jensen-Shannon Divergence
D_{KL}	Kullback-Leibler Divergence
s_i	Surprise-score for node v_i
r_i	Interest-score for node v_i
$\hat{S}_a$	Surprise scores for all neighbors of v_a
$\hat{R}_a$	Interest scores for all neighbors of v_a
w_s, w_r	Weights when s_i and r_i are combined
f_j	j-th feature for nodes
λ_j	Weight of feature f_j
$L_{i,j}$	Neighborhood dist. of node v_i for feature f_j
G_j	Global distribution for feature f_j
U_j	User profile distribution for feature f_j

3.3 Problem Definition

The input is a graph $G = (V, E, A)$ where A is a set of attributes, V the vertices, and E the edges. Each node $v_i \in V$ has a corresponding attribute value for each attribute (feature) $f_j \in A$ (e.g., degree). Our approach works with both numerical and categorical attributes. We assume there are no self-loops (i.e. edges connecting a node to itself). We solve the following problem with FACETS:

Definition 1 ***Node-Sequence Aware Ranking.*** *Given a starting node v_a, a sequence of nodes $V_h \subset V$ in which a user has shown interest, how can we find the top-k nodes among the neighbors of v_a that are (1) similar by features to the sequence of V_h nodes (subjective interest) and (2) uncommon compared to the global distribution (surprising or unexpected)?*

Graph exploration is an interactive and iterative process, where the user incrementally explore parts of the graph. FACETS solves the above problem.

A common approach to rank nodes is by their *importance* scores, which are often com-

puted using PageRank [158], Personalized PageRank [159] or random walk with restart [160]. However, there are other ways to rank the nodes, like using surprise or interest [161, 157]. We have chosen to rank nodes by their surprise and user-driven interest rather than by the more conventional importance metrics. We chose *surprise*, because serendipitous results and insight do not always come from the most topologically important nodes [161]. We made FACETS adaptive, because what makes nodes interesting varies from person to person. For each node we suggest a combination of the most surprising and most interesting neighbors at each step of the journey.

3.3.1 Feature Distributions

FACETS uses feature-based surprise and interest in order to guide the graph exploration process. Even when a dataset does not contain node-level features, we can derive node features by using common graph-centric measures like PageRank, centrality measures or labels drawn from clustering (community detection) approaches. This means that even without a set of initial features, it is still possible for FACETS to guide graph exploration.

FACETS requires a compact representation of feature distributions. Histogram is a natural and computationally inexpensive way to represent distributions. Our approach can consider any histogram, regardless of the binning strategy — e.g., equi-width or equi-height binning — used to infer the histogram. Here, we opt to use the parameter-free technique by Kontkanen and Myllymaki [162] that is based on the Minimum Description Length (MDL) principle. In a nutshell, it identifies as the best binning one that best balances the complexity of the histogram and the likelihood of the data under this binning. In practice this means it automatically chooses both the number of and locations for the cut points, that define the histogram. It does so purely on the complexity and size of the data.

Definition 2 ***Representing Local Feature Distributions.*** *We first create a histogram for a given feature f_j and a set of nodes V with their feature values. A histogram consists a set of bins $b \in B_j$ each of which has a probability value based on the number of nodes*

corresponding to. Although we have chosen MDL binning to construct our histograms, FACETS *will work with most histograms and binning approaches.*

The ***neighborhood (or local) distribution*** $L_{i,j}$ *is a distribution of features* f_j *over a set of neighbors of a particular node* v_i *(visualized in* FACETS *in Figure 3.2.c in orange); The* ***global distribution*** G_j *is the feature distribution across all nodes (visualized in* FACETS *in Figure 3.2.c in gray); and the* ***user profile distribution*** U_j *is the distribution of a sequence of interesting nodes* V_h *collected from the user during interaction with* FACETS.

FACETS works by guiding users during their graph exploration using both surprisingness and subjective interest that changes dynamically to suit the user. We do each of these rankings by comparing the local or neighborhood feature distributions with the global to determine surprisingness and the local with a user profile to determine dynamic subjective interest.

3.3.2 Ranking by Surprise

In order to calculate a node's surprisingness we compare the distribution of the node's neighbors with the global distribution for each feature. We chose a combined feature-centric and structural approach, because both structure and features play a critical role in

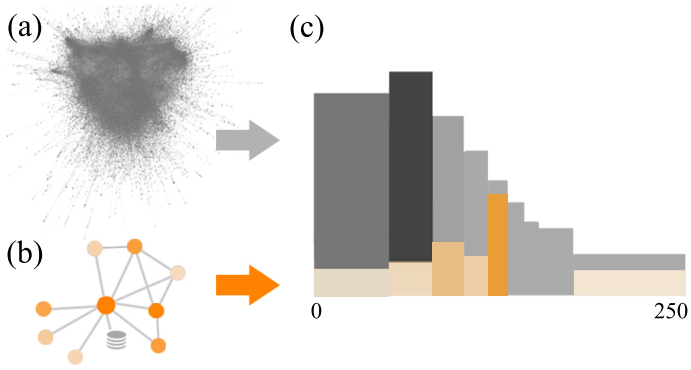


Figure 3.2: Local and global distribution histograms are both essential to FACETS. Local histograms (orange bars) are representations of feature distributions in a single node's egonet. The global distributions (gray bars) depicts the corresponding feature's distribution across the whole graph. The difference between those two distributions is an indicator of whether or not a node is "unexpected" or surprising compared to the majority in the graph.

inference problems [163]. Nodes whose local neighborhood vary greatly from the global are likely to be more surprising as they do not follow the general global trends.

One approach is to use the base entropy over node features to detect anomalous nodes; however, this ends up biasing the ranking towards a skewed distribution. Instead we measure the difference between two distributions for more consistent results.

Through our experiments we have chosen Jensen-Shannon (JS) divergence, a symmetrical version of Kullback-Leibler divergence to construct our surprisingness metric. JS divergence works well, because the resulting output is in bits so the divergences of several features can be easily combined into a single score. We measure surprise by determining the divergence of feature distributions $L_{i,j}$ over a node's neighborhood V_a (1 hop), from the global distributions of features G (see Equation 3.3). From these scores we select the top-k most surprising nodes (Equation 3.4).

Given the JS Divergence or information radius between two distributions P and G:

$$D_{JS}(P||G) = \frac{1}{2}D(P||Q) + \frac{1}{2}D(G||Q), \tag{3.1}$$

where $Q = \frac{1}{2}(P + G)$ and $D(P||G)$ is the KL divergence for discrete distributions:

$$D(P||G) = \sum_{b} P(b) \log \frac{P(b)}{G(b)} \tag{3.2}$$

In Equation 3.2 we use base 2 so that $0 \leq D_{JS}(P||G) \leq 1$. For a fresh node v_a, whose neighbors are not yet visualized we first compute the surprise-score, s_i, of all neighboring nodes $v_i \in N(v_a)$:

$$s_i = \sum_{f_j \in A} \lambda_j D_{JS}(L_{i,j}||G_j), \tag{3.3}$$

where L_j and G_j are the local and global distributions of node-feature f_j and λ_j is a feature weight. Weighted feature scores in Equation 3.3 are used to lessen the impact of noisy features and to allow the user to lessen the contribution of a feature manually. The s_i

scores are composed into $\hat{S}_a$, which holds all the scores for the neighbors of initial node v_a. We can find the most surprising k-nodes by looking for the largest divergence from the global:

$$\arg\max_{1...k} \hat{S}_a \tag{3.4}$$

This yields the top-k most surprising nodes among the neighbors of node v_a. Since both the local-neighborhood feature distributions and the global feature distributions are static, the surprise scores can be precomputed to improve real time performance. We precompute and store surprise in FACETS to improve performance.

3.3.3 Ranking by Subjective Interestingness

We track the user's behavior and record a user profile as they explore their data. Each clicked node offers valuable details into the types of nodes in which the user is interested. This forms distributions U_j for each feature f_j.

To rank the user's interest in the undisplayed neighbors of node v_a we follow a similar approach as Equation 3.3:

$$r_i = \sum_{f_j \in A} \lambda_j D_{JS}(L_{i,j}||U_j), \tag{3.5}$$

where U_j is the distribution of feature f_j from the user's recent node browsing. In this case we want the local distributions that match better the user's current profile; i.e. we want the smallest possible divergences:

$$\arg\min_{1...k} \hat{R}_a \tag{3.6}$$

Since this suffers from the cold-start phenomenon, because a user will not have a profile until they have explored some nodes, our remedy is to simply start the suggestions with surprising and important nodes, until the user has investigated several nodes.

In this section we cover the graph visualization and design of FACETS.

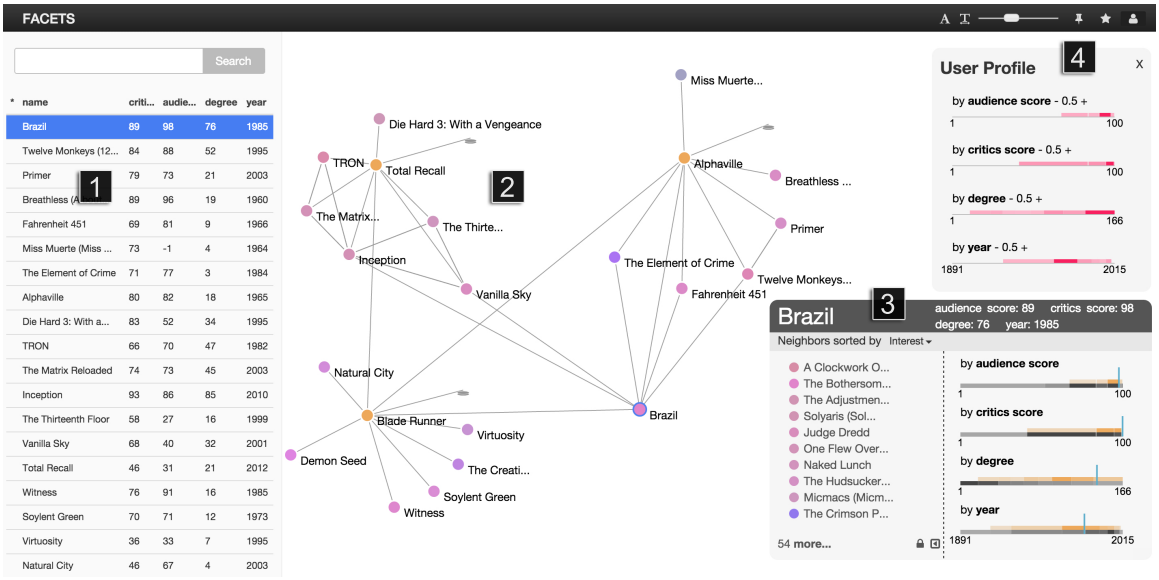


Figure 3.3: The FACETS user interface displaying an explored portion of the RottenTomatoes similar-movie graph. The user has traversed several films (shown as orange nodes with titles) and FACETS has displayed a subset of the relevant neighbors (the nodes with colors ranging from blue to red) and their connectivity. 1: The *Table View* provides a conventional summarization of the already explored nodes and some of their features. 2: The *Graph View* shows the connectivity of similar films as the user explores (it is linked with the table so that a selected node is highlighted in both views. 3: The *Neighborhood Summary* shows the currently hidden nodes, by their feature distributions, for a selected film. 4: The *User Profile* demonstrates a heat-map or flat histogram of the features the user has covered so far in their exploration. figures/adaptivenav are viewed best in color.

3.4 Components

We have created an adaptive graph exploration tool, FACETS, for performing fast and intuitive exploration of graph datasets. FACETS was designed especially for graph tasks that require node-level details.

FACETS's user interface as shown in Figure 3.3 has four key elements: The first main area is the **Table View** (1) showing the currently displayed nodes and their features. This provides sortable node-level information. The central area is the **Graph View** (2). It is an interactive *force-directed graph layout* that demonstrates the structure and relationships among nodes as the user explores. Node colors are used to encode the surprise and interest based on the user's current exploration. We have the **Neighborhood Summary** (3) to

summarize neighbors, as we cannot show a large number of neighbors in the Graph View. The neighborhood summary allows a user to investigate the feature distributions of its currently undisplayed neighbors as well as sort them by their interest or surprise scores.[2] It presents the user with feature *heat maps*[3] that summarize the distributions of hidden nodes. When clicked, the heat maps turn into conventional distribution plots (histograms), where a user can compare the local neighborhood (orange) and the global (gray). This lets a user quickly select new nodes based on their feature values and get a quick summary of this node's neighborhood. As a user explores, we construct and display a summary profile of the important features they have covered in the **User Profile** view (4). The user profile view suggests high-level browsing behavior to the user; allows for better understanding of where the user-interest ranking comes from; and allows them to adjust if they want to ignore certain features in the interest ranking.

3.4.1 Design Rationale

In the following paragraphs, we discuss our design rationale.

Exploring and Navigating

One of our design goals is to facilitate both exploration and navigation of graphs. We use the term *graph navigation* to refer to the act of traversing graph data with a known destination or objective. *Graph exploration* is more like foraging through the graph without a particular destination. We facilitate navigation through adaptation and exploration by filtering out unsurprising and unimportant nodes while still providing crucial feature details for hidden nodes via the *Neighborhood Summary* window. As shown in Figure 3.4, the user can bring up a summarized view of mouse-hovered nodes where the top ranked hidden

[2] FACETS focuses on novel ranking measures. Conventional measures (e.g., degree, PageRank, etc.) could be supported as additional ranking choices in the drop-down menu in Figure 3.3.3.

[3] While histogram encodes value of each bin as height, heat map uses darkness to represent values with equal height. The main advantage of a heat map over histogram is its compact representation, which helps us save space.

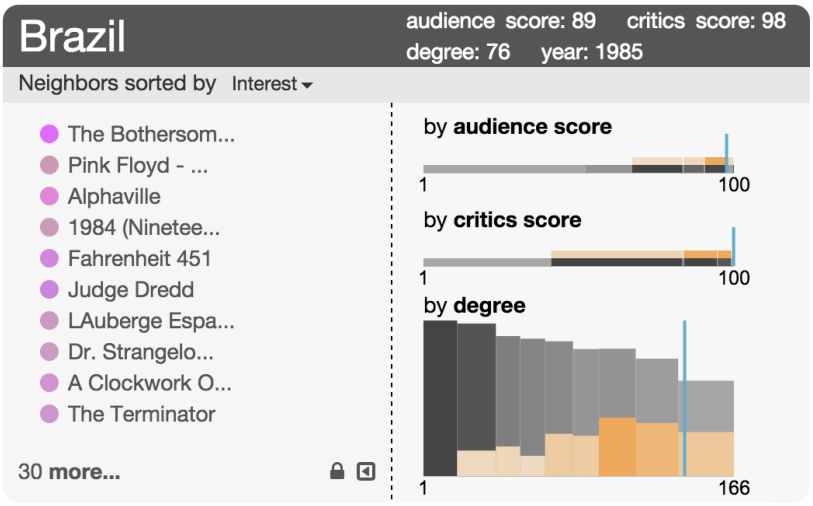


Figure 3.4: FACETS's neighborhood summary view, which displays the top ranked neighbors of the given node (left) and distributions of the current neighborhood's features (right). Each feature is displayed by a compact heat map, which can be expanded into a histogram. Each heat map shows both the global distribution (gray) and the node's neighborhood's distribution (orange).

neighbors, local distribution and global distribution are displayed. These neighborhood feature distributions allow quick and easy filtering.

Show the Best First

Keeping the graph view from becoming an incomprehensible mess of edges means only showing relevant, surprising, and interesting nodes. Importance, surprise, and user-interest are all important aspects of discovery, so we blend them into the results that are shown first to the user. Figure 3.5 illustrates how we visually encode the interest-surprise difference by hue and the sum of both scores by saturation. Nodes ranked high tend to have brighter color closer to purple, which becomes a clear visual cue for the user to quickly identify desired nodes. FACETS is almost completely free of parameters, making it simpler for users to explore their graphs.

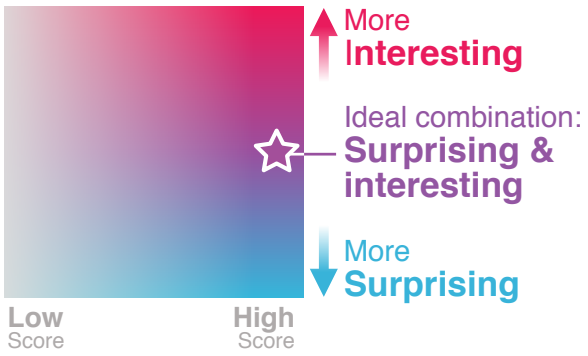


Figure 3.5: The colors used to encode surprise (blue) and interest (pink). The surprise and interest scores are not mutually exclusive so ideal candidates may be both surprising and interesting (purple).

Adaptive and Adjustable

Because user-interest varies greatly across users and even time, our design must be able to track the user's exploration behavior in order to approximate what is motivating them. Adapting as the user explores helps provide critical insight into users' latent objectives, because they can see how they have explored and also may find what they seek. During exploration, the users profile updates dynamically to illustrate a summary of their feature traversal, while the graph view provides the topological traversal. It is not necessary to preset any parameters in order for our adaptive algorithm to work, because the rankings are done in a black-box fashion during users' explorations. We allow them to directly manipulate the balance of features used in the interest calculation and choose which features form the ranking. This enables the user to dynamically increase or decrease the importance of any features during their exploration and immediately impact the interest ranking.

3.5 The FACETS Algorithm

In this section, we summarize the process of finding top-k most interesting and surprising neighbors in Algorithm 3.6. Whenever a user selects a node to explore, we rank its neighbors based on surprise and subjective interestingness we explained in the previous section. For each of the neighbors, we compute surprise and interest scores for each feature and

aggregate them based on feature weights λ_j. We blend those scores, and return the k nodes with highest scores.

---

procedure RANKNEIGHBORS(node v_a, precomputed histograms for: global feature distributions G and user profile distributions U, feature weights λ_j, weights for surprise w_s and interest w_r)
  if deg(v_a) $\geq$ 10000 **then**
    $V_a \leftarrow$ top 10000 neighbors of v_a by highest degree

  else
    $V_a \leftarrow$ all neighbors of v_a
  end if
  for all nodes v_i in V_a **do**
    for all features f_j in A **do**
      $s_i^{(j)} = D_{JS}(L_{i,j}||G_j)$ ▷ see Eq. 3.3
      $r_i^{(j)} = D_{JS}(L_{i,j}||U_j)$ ▷ see Eq. 3.5
      $t_i^{(j)} = w_s s_i^{(j)} + w_r(1 - r_i^{(j)})$
    end for
    $\hat{T}_a[i] = \sum_{f_j \in A} \lambda_j t_i^{(j)}$
  end for
  $T_{Nodes} = \arg\max_{1...k} \hat{T}_a$
return T_{Nodes}
end procedure

Figure 3.6: RankNeighbors

We evaluate the effectiveness and speed of FACETS using large real-world graphs. FACETS is designed to support open-ended discovery and exploration suited to users' subjective interests, which is inherently challenging to evaluate [164]. Traditional quantitative user studies (e.g., measuring task completion time) would impose artificial constraints that interfere with and even potentially suppress how users would naturally explore based on curiosity, counteracting the benefits that FACETS aims to foster. Given the exploratory nature of FACETS, canonical quantitative metrics of "success" like precision, recall, MAE, and RMSE [165, 166, 167] are not directly applicable here. For these reasons, we demonstrate FACETS's effectiveness through several complementary ways: (1) a small observational study based on the study of exploratory systems from [164], (2) run time analysis of our surprise and interest rankings on four real world graphs, (3) a comparison of our scoring

with canonical node ranking techniques, and three case studies that investigate the results of our algorithm on a movie graph and citation network.

3.5.1 Graph Datasets

We use the Rotten Tomatoes[4] (RT) movie dataset as our main dataset, which is an attributed graph that contains basic information per movie (e.g., released year), as well as users' average ratings and critics scores. The observational study used only the RT graph. For the analysis of runtimes, we also used the Google Web network, the DBLP co-authorship graph, and the YouTube network datasets from the SNAP repository [168]. Table 3.2 shows the graphs' basic statistics and in which parts of our evaluation they were used.

3.6 Observational Study

We conducted a small observational study with semi-structured interviews and surveys. Four participants were recruited through Georgia Institute of Technology mailing lists. We screened for participants with at least basic knowledge of movies (e.g., enjoy occasionally watching movies when they grow up). Three subjects were female and one was male, all had completed a bachelor's degree. They ranged in age from 21 to 27, with an average age of 23.

The participants were provided a 10-minute tutorial of FACETS, which demonstrated the different parts of FACETS and how they can be used to investigate the RT graph. They were asked to think aloud for the whole study, so that if they became confused or found something interesting we would be able to take notes. For all tasks, participants were free to choose movies to inspect, so that they would remain interested during their exploration. They could also look up movies on RottenTomatoes' website if they were curious about details.

Every participant performed three general tasks, each lasted for 10 minutes:

[4] `http://rottentomatoes.com/`

Table 3.2: Graph datasets used in our observational study, speed testing, and case studies. They were picked for their variety in size and domain. Rotten Tomatoes was used in the observational study due to its general familiarity to the public.

Network	Nodes	Edges	Obs. Study	Speed	Case Study
Rotten Tomatoes	17,074	72,140	✓	✓	✓
DBLP	317,080	1,049,866		✓	✓
Google Web	875,713	5,105,039		✓	
Youtube	1,134,890	2,987,624		✓	

1. Open exploration of the RT graph to help acquaint the participants with FACETS
2. Investigation of the surprising neighbors of movies using the neighbor summary view (participants chose their own starting movies)
3. Exploration of movies, chosen by the participant, with consistent years (e.g., around mid 90's)

The first task was presented in order to encourage the participants to ask questions about the system, as well as investigate how they would use it without being directed. We were curious about which features they would use and if there were any behavioral patterns we could find during exploration.

The second task was used to investigate quality of the surprising results. We let the participants pick their own starting movies since it would be easier for them to work with movies they knew. Our requirement was that the movie had at least five neighbors, so that the exploration options weren't trivial.

For the third task, we asked participants to choose and investigate a set of movies that interest them, with consistent years, so that they could see an example of how FACETS will adapt the interest ranking based on their recently clicked nodes. This task allowed the participants to comment on and better understand the interest ranking, and allowed us to get feedback on the quality of subjectively interesting results. The observations and feedback from the study allowed us to understand how FACETS's visual encoding guides participants during exploration.

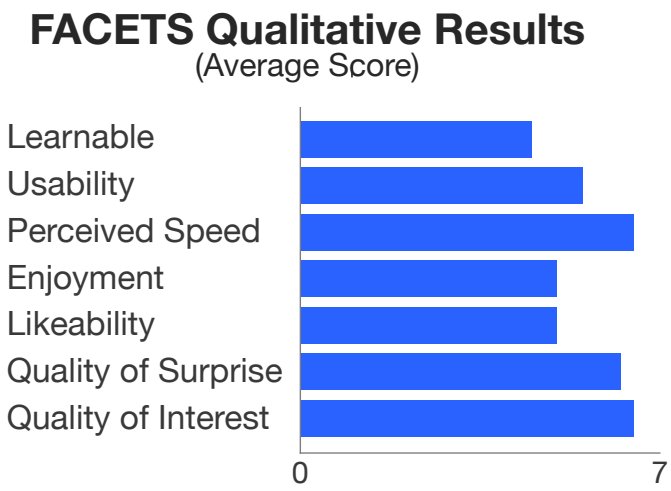


Figure 3.7: The average qualitative responses for FACETS.

3.6.1 Observational Study Results

We measured several aspects of FACETS using 7-point Likert scales (provided as a survey at the end of the study). The participants enjoyed using FACETS and additionally found that our system was easy to learn, easy to use and likeable overall; although this is a common experimental effect, we find the results encouraging (shown in Figure 3.7). Users found both our rankings to be useful during their exploration (see the last two bars in Figure 3.7). Several participants stated that the visualization combined with the interest and unexpectedness rankings to be very exciting during exploration. One participant stated, *"it was exciting when I double clicked a node and saw how it was connected to my explored movies"*. FACETS was able to find subjectively interesting content for our participants as they explored.

Participants primarily spent time in two areas, on the main graph layout and in the neighborhood summary view. They used the neighborhood summary view to find and add new nodes and then inspected the relationships of these newly added nodes in the graph view. The table view was used primarily to select already-added nodes by name.

Two participants reported not fully understanding how the user profile was affecting the results until the second task. The participants used a common strategy during exploration,

in which they would add neighbors to a desired node and then spatially reorganize the results by dragging some of the new nodes to a clear area. They repeated this process and often inspected new nodes that shared edges with previous content.

In summary, our design goal was generally met: our participants deem FACETS as an easy-to-learn and easy-to-use system with highly rated qualities of both interesting and unexpected neighborhood suggestions.

3.6.2 Runtime Analysis

Next, we evaluate the scalability of FACETS over several million-edge graphs (Table 3.2). Our evaluation focuses on demonstrating FACETS's practicality in computing exploratory rankings in time that is linear in the number of neighbors and of node attributes, returning results in no more than 1.5 seconds for the 5 million edge Google Web graph that we

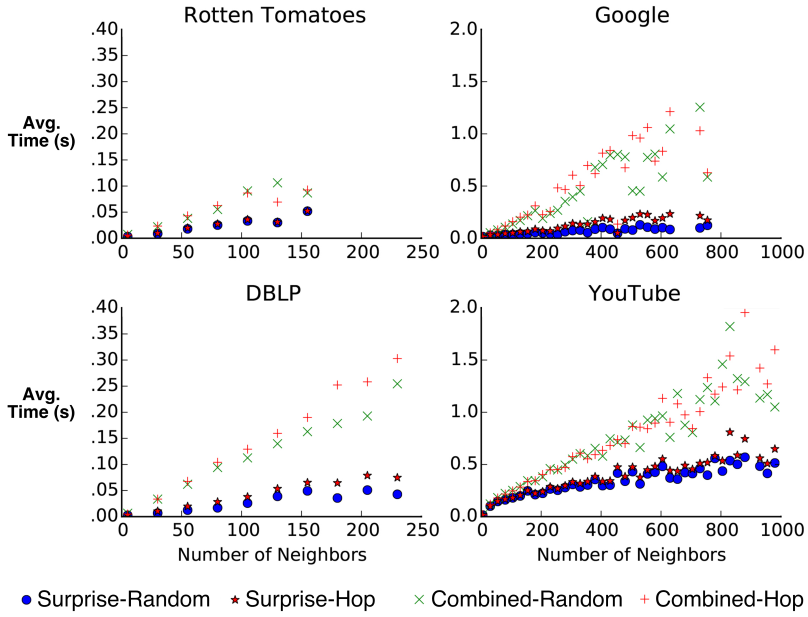


Figure 3.8: FACETS ranks neighbors in linear time, which is necessary to handle the large numbers of nodes that a user may explore. We show the average time to calculate the JS divergence for surprise and the combination of surprise and interest over a neighborhood of size n. FACETS combines the ranks if there is sufficient user profile data. We tested with contiguous node ordering to simulate normal exploration and random ordering to simulate a user searching using the table view.

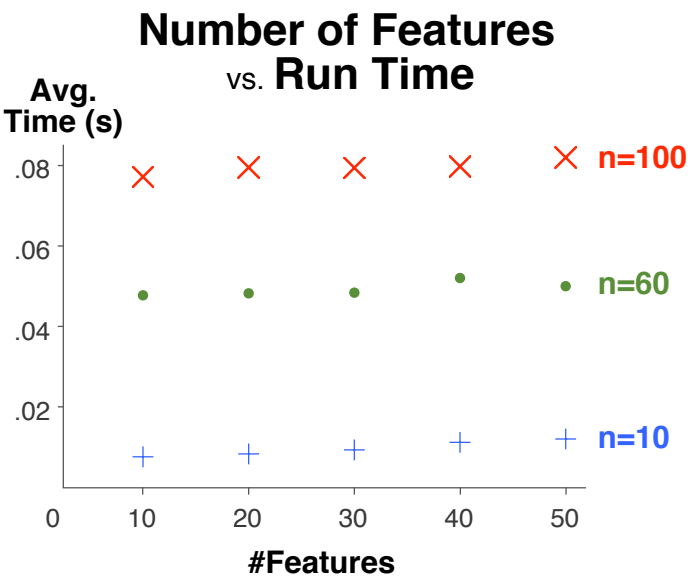


Figure 3.9: FACETS scales linearly in the number of features (here n tracks three sizes of neighborhood).

tested. We expect these runtime results will significantly improve with future engineering efforts and optimization techniques. The experiments were run on a machine with an Intel i5-4670K at 3.65 GHz and 32GB RAM.

One of our goals is sub-second rankings, so that interactions with FACETS are smooth. This is why we have chosen to treat nodes in the tail of the degree distribution separately than their modest degree neighbors.

We have analyzed the runtime of FACETS, in Figure 3.8, using the graphs from Table 3.2; all but the RT graph used eight synthetic features. We use both random ordering and contiguous node ordering, displayed as Rand and Hop in Figure 3.8. Random ordering simulates using the search functionality while hop ordering simulates hopping from one node to its neighbors during exploration. High degree nodes have a higher chance of being selected and account for the fact that hop sometimes is slower than random in Figure 3.8. The graphs we tested demonstrate that the cost of the ranking is linear in the number of neighbors in the neighborhood. Our ranking requires both a value lookup and a single JS divergence calculation for each node and for each feature.

As mentioned earlier, the surprise scores are precomputed and can be accessed quickly. The cost to rank neighbors comes largely from the interest scores which cannot be precomputed. The cost, in JS divergence calculations, is $O(n \cdot f)$, and is asymptotically linear in both the number of neighbors n and the number of features f. Given our use of MDL histograms for the features, we can scale the number of features at low linear incremental cost (see Figure 3.9). Each neighbor only requires exactly one JS divergence calculation per feature (comparing the user profile and the local distribution).

Since many graphs contain triangles, it is very likely that redundant calls will be made during a user's exploration. We use this to our advantage and *cache* the distributions for each visited node rather than re-fetching them each time. Graphs with higher clustering coefficient may achieve better caching performance. For all but the YouTube graph, the caching became memoizing as the entirety of the nodes could fit in the cache.

3.7 Case Studies

Here, we describe three case studies using two graph datasets — the first two using the Rotten Tomatoes (RT) movie graph, and the last one using the DBLP co-authorship graph. These case studies illustrate how FACETS helps the user explore graphs incrementally, gain understanding, discovers new insights, and visualize them. FACETS adapts to the user to provide surprising and interesting rankings to help the user explore relevant parts of the graphs.

3.7.1 Movie Example I: Blade Runner

Our first user John likes mid-90's post-apocalyptic and action films (shown as gray circles in Figure 3.10, e.g., *The Crow*, *Waterworld*), many of which were well received by critics and audience alike. John can add more films that he likes by exploring node-to-node or by the search utility. After exploring a few films in such genre, John is particularly interested in **Blade Runner**. Based on what John has explored, FACETS returns top-ranking

surprising and interesting films. The top-5 of each kind are displayed in Table 3.3 on the left and in Figure 3.10. These movies' interest (I) and surprisingness (S) scores are determined based on conventional measures of node importance; PageRank (PR), betweenness centrality (BC), and eigenvector centrality (EV).

Many of the surprising movies would not considered important by canonical approaches, partly because FACETS's surprise rank operates on both features and local graph structure rather than global structure. Movies that are very heavily connected also face a higher chance of matching the global distribution and therefore being less surprising.

While the notion of surprise is often considered "unimportant" by conventional metrics, interest exhibits more variety in importance than surprise. This is especially apparent in both this and the next case studies. Here we have selected types of films that have really large viewership and represent a very large genre in modern film with many potentially similar movies. Both the structure and features used in our subjective interest have lead the ranking towards more conventionally important nodes (consider the high PR, BC, and EV scores).

As the highest possible JS-divergence for a single feature is 1; the maximum divergence is the sum of feature weights Λ. In this case study, we used five features, so interest scores ≤ 1.0 suggest that the user profile is in relatively good agreement with the proposed node. Nodes with very low interest divergence are strong candidates that their neighborhood will be of interest to the user.

Table 3.3: Comparing FACETS's surprise and interest ranking with common importance rankings (I&S is interest or surprise, PR is PageRank ($\times 10^{-4}$), BC is betweenness centrality, EV is eigenvector centrality ($\times 10^{-3}$). Each example has a selected film, a user profile at the time of selection, and the interesting and surprising neighbors.

		Example I			
Current Movies		**Blade Runner**			
Movies Visited		Waterworld, Braveheart, Pulp Fiction, The Crow, Fargo			
Top-k by	**Title**	**I**	**PR**	**BC**	**EV**
Interest	L.A. Confidential	1.06	2.23	264k	7.10
	Sin City	1.15	4.34	833k	44.7
	Dredd	1.20	1.74	55k	43.3
	V for Vendetta	1.23	2.99	620k	43.5
	Heat	1.23	4.18	385k	11.0
Top-k by	**Title**	**S**	**PR**	**BC**	**EV**
Surprise	The Creation of the Humanoids	2.93	0.19	851	0.58
	Demon Seed	2.75	0.50	650	1.54
	Natural City	2.69	0.28	437	0.74
	Virtuosity	2.64	0.35	668	1.61
	Soylent Green	2.59	0.57	6578	1.50

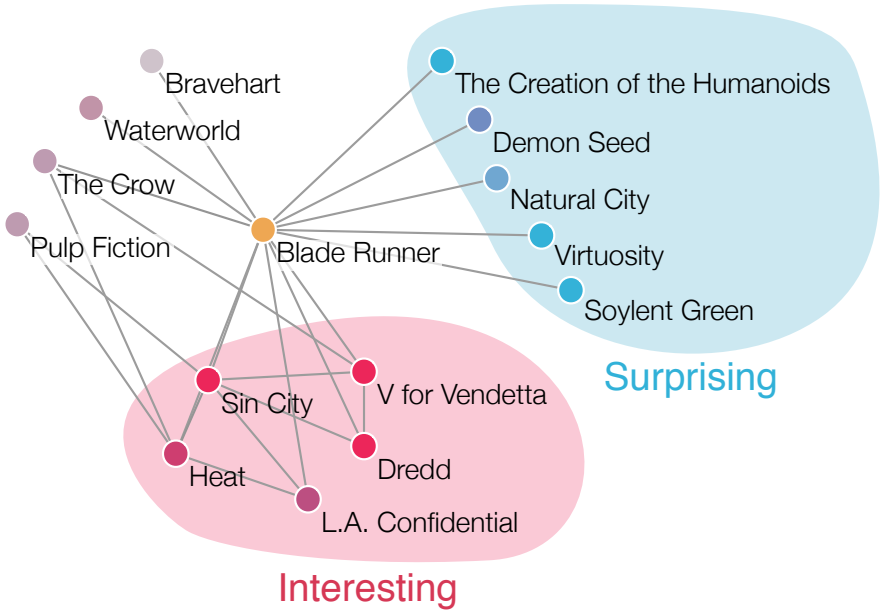


Figure 3.10: Visualizations of the **Blade Runner** case study. FACETS shows the main anchor node in yellow and the neighbors colored according the surprise and interest using the scale from Figure 3.5.

3.7.2 Movie Example II: Toy Story

Our second user Bonnie investigates several children's films that were criticized by the critics, but still enjoyed by audiences (all have a lower critics' score than audience score). Bonnie hops from node-to-node across the listed films in the order they appear in Table 3.4. She then selects **Toy Story**. The results are displayed in Table 3.4 on the right and in Figure 3.11.

As in the previous example, the surprising nodes tend not to be conventionally important. Yet, consider the difference in importance of the interesting films versus the previous example, many of these suggestions have significantly lower importance. The second profile has less coherent features and they do not draw the the interest ranking towards conventionally important nodes, despite that *Toy Story* is a very well connected node. Also note that **Monsters University** and **ParaNorman** are featured in both the top-5 surprising and interesting movies. The surprise score and interest score are not counter to each other. In

Table 3.4: Comparing FACETS's surprise and interest ranking with common importance rankings (I&S is interest or surprise, PR is PageRank ($\times 10^{-4}$), BC is betweenness centrality, EV is eigenvector centrality ($\times 10^{-3}$). Each example has a selected film, a user profile at the time of selection, and the interesting and surprising neighbors.

	Example II				
Current Movies	**Toy Story**				
Movies Visited	A Bug's Life, Kung Pu Panda, Jumanji, The Incredibles, How to Train Your Dragon				
Top-k by	**Title**	**I**	**PR**	**BC**	**EV**
Interest	Monsters University	0.54	0.76	1070	29.3
	Rio	0.63	1.66	25464	57.4
	Toy Story II	0.70	1.84	55486	59.1
	The Iron Giant	0.71	1.43	54079	46.3
	ParaNorman	0.71	0.94	23291	15.1
Top-k by	**Title**	**S**	**PR**	**BC**	**EV**
Surprise	Buzz Lightyear of Star Command	3.15	0.14	12	2.14
	Small Fry	3.01	0.17	132	3.10
	Monsters University	2.64	0.76	1070	29.3
	Cloudy With a Chance of Meatballs	2.46	1.28	1027	42.5
	ParaNorman	2.45	0.94	23291	15.1

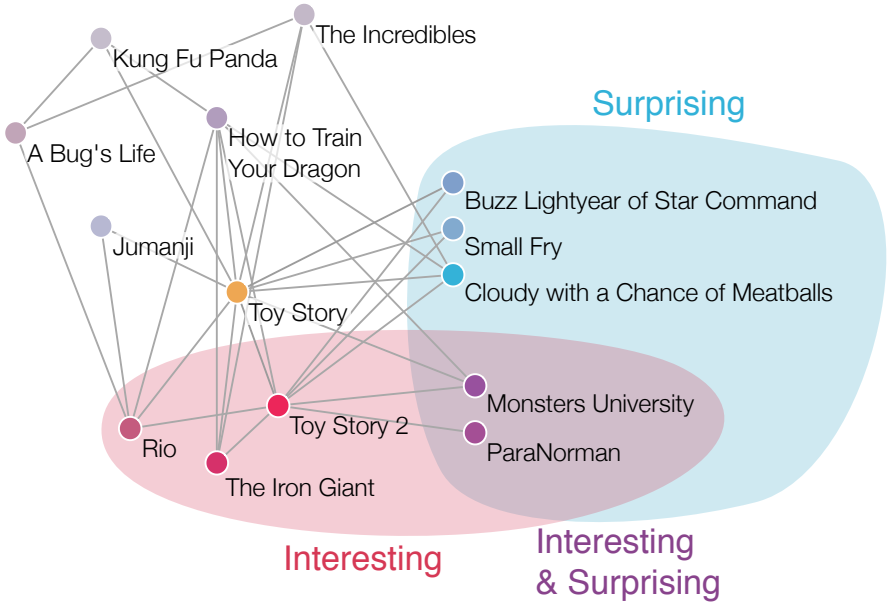


Figure 3.11: Visualizations of the **Toy Story** case study. FACETS shows the main anchor node in yellow and the neighbors colored according the surprise and interest using the scale from Figure 3.5

fact, this is an excellent example, because these films are both surprising and subjectively interesting!

3.7.3 DBLP Example: Data mining and HCI researchers

Our third example uses data extracted from DBLP, a computer science bibliography website [5]. The graph is an undirected, unweighted graph describing academic co-authorship. Nodes are authors, and an edge connects two authors who have co-authored at least one paper.

Our user Jane is a first-year graduate student new to data mining research. She just started reading seminal articles written by Philip Yu (topmost orange node in Figure 3.12). FACETS quickly helps Jane identify other prolific authors in the data mining and database communities, like *Jiawei Han*, *Rakesh Agrawal*, *Raghu Ramakrishnan*, and *Christos Faloutsos*; these authors have similar feature distributions as Philip Yu (e.g., very high degree). Jane chooses to further explore *Christos Faloutsos*'s co-authors. FACETS suggests

[5] http://dblp.uni-trier.de

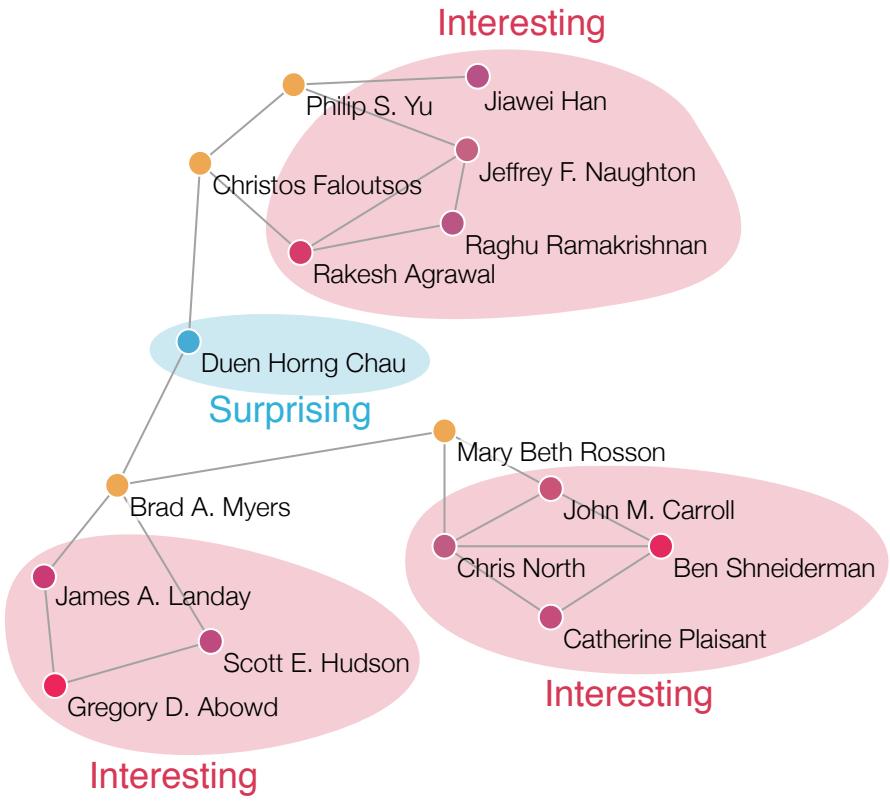


Figure 3.12: Visualization of our user Jane's exploration of the DBLP co-authorship graph. Jane starts with Philip Yu. FACETS then suggests Christos Faloutsos and several others as prolific data mining researchers. Through Christos, FACETS suggests Duen Horng Chau as a surprising author as he has published with both data mining and human-computer interaction (HCI) researchers, like Brad Myers. Through Brad, FACETS helps Jane discover communities of HCI researchers, including Ben Shneiderman, the visualization guru.

Duen Horng Chau as one of the surprising co-authors, who seems to have relatively low degree (i.e., few publications) but has published with highly-prolific co-authors. Among these is *Brad Myers* (leftmost orange node in Figure 3.12), who publishes not in data mining, but in human-computer interaction (HCI). This exploration introduces Jane to a new field, and she wants to learn more. Using FACETS's interest-based suggestion, she discovers a community of co-authors who have published with Brad; among them, *Mary Beth Rosson* further leads to another community of HCI researchers, which includes *Ben Shneiderman*, the visualization guru!

In this chapter, we presented FACETS, an integrated approach that combines visualiza-

tion and computational techniques to help the user perform adaptive exploration of large graphs. FACETS overcomes many issues commonly encountered when visualizing large graphs by showing the user only the most subjectively interesting material as they explore. We do this by ranking the neighbors of each node by surprisingness (divergence between local features and global features) and subjective interest based on what the user has explored so far (divergence between local features and user profile).

Our FACETS algorithm is scalable and is linear in the number of neighbors and linear in the number of features. We evaluated FACETS with a small observational study, wherein participants consistently rated FACETS well. We demonstrated the effectiveness of FACETS through case studies using the Rotten Tomatoes movie graph and the DBLP co-authorship graph, and comparison with canonical importance ranking measures. Despite the old adage that you can't see the forest for the trees, with FACETS you can see the graph through its nodes.

Thrust II

Constructing, Refining, and Performing Graph Queries

OVERVIEW

By utilizing techniques, approaches, and the FACETS system in Thrust I: Local Graph Exploration, an analyst can quickly collect numerous interesting patterns and subgraphs from their network. Other exact matches or similar subgraphs may exist, but be topologically far from the currently explored region. Manually scouring for these would be a challenging and tedious task.

How can they locate more of these patterns or find similar matches in the rest of their network? The data mining techniques of subgraph matching and graph querying offer an automated solution. Graph querying improves the reach of local exploration once a pattern is known (or even partially known).

Currently graph querying requires complex querying languages; however, we show that queries can instead be formed via a visual, interactive system. Querying is often an iterative process [8, 9] that can benefit greatly from visual aids. Making such a system requires both algorithms and visualization working in tandem, as few algorithmic approaches have been designed for both interaction and visualization. This Thrust focuses on our techniques, approaches, and systems that: help analysts visually construct and refine queries; and algorithms to use these queries to seek out subgraphs of interest from the network.

- **MAGE (Chapter 4)** a scalable, multicore subgraph matching approach that supports expressive queries over large, categorically-attributed graphs.

- **VISAGE (Chapter 5)** an interactive visual graph querying approach that empowers analysts to construct, refine and dispatch expressive queries, without writing complex code.

CHAPTER 4

MAGE: MATCHING APPROXIMATE PATTERNS IN RICHLY-ATTRIBUTED GRAPHS

Given a large graph with millions of nodes and edges, say a social network where both its nodes and edges have multiple attributes (e.g., job titles, tie strengths), how to quickly find subgraphs of interest (e.g., a ring of businessmen with strong ties)? We present MAGE, a scalable, multicore subgraph matching approach that supports expressive queries over large, richly-attributed graphs. Our major contributions include: (1) MAGE supports graphs with both node and edge attributes (most existing approaches handle either one, but not both); (2) it supports expressive queries, allowing multiple attributes on an edge, wild-cards as attribute values (i.e., match *any* permissible values), and attributes with continuous values; and (3) it is scalable, supporting graphs with several hundred million edges. We demonstrate MAGE's effectiveness and scalability via extensive experiments on large real and synthetic graphs, such as a Google+ social network with *460 million* edges.

4.1 Introduction

Graphs are a convenient and ubiquitous means to represent many naturally occurring patterns. Rich with information, many graphs contain subgraph patterns that capture interesting dynamics among entities, but because of the size and complexity of these graphs, spotting such interesting patterns can be a difficult task. For instance, an analyst may want to better understand the inner working of criminal activities by analyzing an intelligence network of various entities (e.g., people or events) connected with edges denoting gathered intelligence (as in Figure 4.1). Sometimes, the analyst may have some initial ideas about how certain suspicious activities may look like. Then, he could use *subgraph match-*

Chapter adapted from work at BigData'15 [Paper Link] [76]

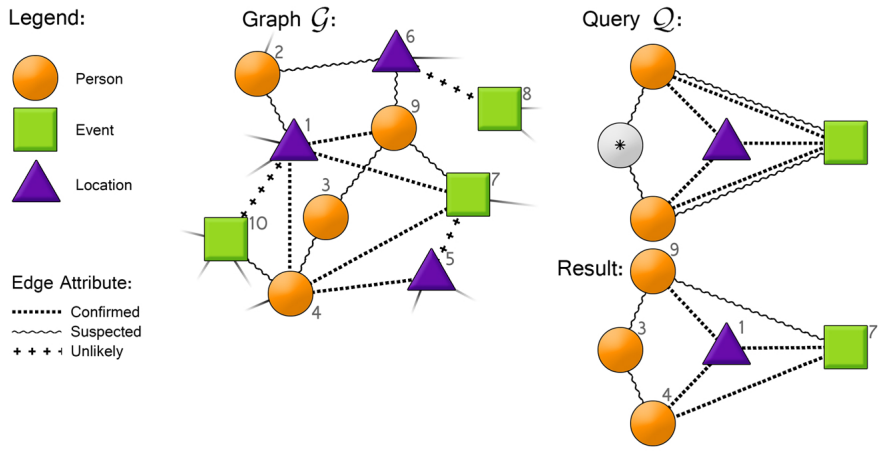


Figure 4.1: An illustrative example showing how MAGE finds patterns in an intelligence graph (middle), which has both node and edge attributes. The node attribute is the entity type, which can be a Person, an Event, or a Location, and the edge attribute is the amount of gathered intelligence for a pair of entities, which can be Confirmed, Suspected, or Unlikely. (Right) A sample query that can be formed in MAGE and a potential result. The query looks for two indirectly related individuals, who were both confirmed at the location of some event and are also believed to have attended the event (denoted by two lines connecting the corresponding nodes). The node labeled with a star indicates a wildcard, which can take any attribute value. A node's identifier is displayed next to it. (All figures are best viewed in color.)

ing techniques to find potentially dangerous individuals. However, he may face multiple technical challenges.

A Motivating Example. Let us consider the intelligence graph in Figure 4.1. Here, the node attribute is the entity type, which can be a Person (orange circle), an Event (green square), and a Location (purple triangle), and the edge attribute is the amount of gathered intelligence for a pair of entities, which can be Confirmed (dotted line), Suspected (wavy line), and Unlikely (line with plusses).

The top right corner of Figure 4.1 shows one query that our analyst may issue, which looks for two indirectly related individuals, who were seen to have appeared at the same location (thus, "confirmed" to be linked to a location) but unclear whether they attended an event together (thus, "suspected" to be linked to an event).

One challenge here is that the analyst might not know what values to assign to some

of the nodes and edges in the query; so instead of guessing or assigning an arbitrary value, the analyst may want to assign a node or an edge with a *wildcard* attribute value, a feature that would allow the analyst to more freely explore different hypotheses. However, it is not supported by most existing subgraph matching techniques.

Another challenge is that the analyst might want to test multiple hypotheses, as in the query in Figure 4.1 where the individuals must be either confirmed *or* suspected of being involved in the event. Rather than forming several distinct queries for each of these values and then combining the results, the analyst should be able to use a single query to retrieve matches that support some, or all, of the multiple hypotheses.

A further challenge is that the analyst may benefit from seeing both exact and *near* matches, because oftentimes an initial pattern may only approximate what an analyst wants to find eventually. Existing subgraph matching approaches that return only exact matches (if any) may not help the analyst evolve his query patterns in this regard. For the query in Figure 4.1, the system should be able to return a match filling in the wildcard, e.g., the person entity corresponding to node 3 in the result. Even with the wildcard, the exact specified structure may not exist in in the graph; under this scenario the system would return a "best-effort" match of the query containing additional nodes and edges. By generating both exact and near matches, we can provide the user with the top-k most closely matched subgraphs even if their initial query was not exactly present in the input graph.

Limitations of Existing Techniques. A representative set of early work on inexact pattern matching on graphs include G-Ray by Tong et al. [75], TALE by Tian and Patel [79], and SIGMA by Mongioví et al. [169]. More recently, Khan et al. proposed the NeMa approach [78], Fan et al. proposed a set of incremental pattern matching algorithms [81, 80] as well as the TopKDiv approach [80], and Cheng et al. proposed the kGPM framework [170]. The main limitations of these techniques are that they either (i) do not support edge attributes, (ii) need computationally-heavy indexes precomputed for the input graph, or (iii) return results that can significantly deviate from the query pattern, which may be difficult

for users to comprehend.

Our Contributions. We present MAGE, a pattern matching system for graphs with node and edge attributes. MAGE, short for ***M**ulti-**A**ttribute **G**raph **E**ngine*, overcomes many of the challenges and limitations outlined above. It produces top-k closest subgraph matches for a large variety of attributed input graphs. We demonstrate in our experimental evaluation that MAGE is a scalable tool that works for graphs from different domains, from intelligence applications to understanding the patterns of movie success.

Specifically, the contributions of this chapter include:

- **Support for node and edge attributes**. MAGE supports graphs with node and edge attributes (categorical or continuous via discretization). Using both node and edge attributes expands the effectiveness of our system on real world data and increases the types of questions that can be answered through graph querying.

- **Flexible queries with rich attributes**. MAGE improves the ease of querying on graphs. In some cases, query information might be limited and the exact attribute of a node or edge may be unknown. MAGE allows the specification of queries with wildcards and across multiple categorical attributes.

- **Fast & scalable algorithm**. MAGE leverages random walk with restart (RWR) steady-state probabilities as proximity scores between nodes to determine how well a subgraph matches the query. We propose a fast and highly scalable multicore approach for RWR calculations. We evaluate MAGE with real and synthetic graphs with several hundred million edges to demonstrate its scalability.

4.2 Problem Definition and Notation

Here, we formalize the subgraph matching problem that MAGE aims to solve.

In its general form, we are given two graphs $\mathcal{G}$ and $\mathcal{Q}$ and we wish to know if $\mathcal{G}$ contains a subgraph that is equivalent to $\mathcal{Q}$. Table 4.1 describes the symbols used in the chapter.

Table 4.1: Symbols and terminology used in this chapter

Symbol	Description
$\mathcal{G}$	$n \times n$ adjacency matrix for $\mathcal{G}$
$\mathcal{A}$	$n \times t$ node-attribute matrix for graph $\mathcal{G}$
$\mathcal{Q}$	Query subgraph to be extracted from $\mathcal{G}$
$\mathcal{G}'$	$(m+n) \times (m+n)$ linegraph-modified bipartite graph
$\mathcal{A}'$	Node-edge-attribute matrix for graph $\mathcal{G}'$
$\mathcal{Q}'$	Query subgraph after edge augmentation
M	A bijective mapping between edges of $\mathcal{G}$ and edge-nodes in $\mathcal{G}'$
$\langle s, t \rangle$	An edge leading from node s to node t
n	Number of nodes in $\mathcal{G}$
m	Number of edges in $\mathcal{G}$
t	Number of distinct categorical attributes
i,j & u,v	Indices of nodes in $\mathcal{Q}$ and in $\mathcal{G}$ respectively.
λ	Ratio of approximated nodes and edges with correct attributes to the total number in a result

This problem is often referred to as the subgraph isomorphism problem. Unfortunately, the subgraph isomorphism problem is NP-Complete [66], making all general solutions computationally infeasible for even modest sized graphs.

The problem we are solving is subtly different than the pure subgraph isomorphism problem. Our problem definition can be formally stated as follows:

Given: (i) A graph $\mathcal{G}$ whose nodes and edges have categorical attributes, (ii) a query graph $\mathcal{Q}$ showing the desirable configuration of nodes connected with edges, each assigned one or more attribute values (or a wildcard), and (iii) the number of desired matching subgraphs k.

Find: k matching subgraphs $\mathcal{Q}_i$ $(i = 1, ..., k)$ that match query graph $\mathcal{Q}$ as closely as possible, according to a goodness metric (which we cover in the Methodology section).

4.2.1 Preliminary: Querying on Node Attributes

Several approaches have been proposed to subgraph isomorphism problem. The work by Tong et al. proposes the G-Ray algorithm [75], which is a best-effort inexact subgraph

matching approach that relies heavily on RWR or personalized page rank values as the selection criterion when constructing query results. This approach uses single nodes as the restarts when calculating the RWR values. We leverage this approach but considerably modify it to allow multiple attributes as restarts in the RWR approximations (see Section 4.4.4 for details). Approximate RWR is still a computationally expensive step that must be performed often. In Section 4.4.1, we show our approach to reducing query latency by decreasing RWR calculation times.

While G-Ray is an integral facet of MAGE, the limitations of the original algorithm are far too constrictive. The G-Ray algorithm is inadequate in supporting expressive querying as it does not support (i) attributed edges, (ii) unknown query attributes, (iii) multiple attributes, and incurs (iv) sizable query latencies on large graphs. These points are what we aim to address with MAGE. Using our linegraph augmentation approach, we support edge attributes, and with wildcards and multiple attributes, we support queries with limited information. To address the speed and scalability we utilize an approximate parallel RWR technique to quickly calculate the data needed to find good candidate matches.

4.3 MAGE Overview

Unlike many previous systems, we focus on both the edge and node attributes in $\mathcal{G}$. We have chosen to use an edge-augmentation method based on the intuition from the linegraph transformation [171]. Under the canonical linegraph transformation, each vertex in the line graph $\mathcal{L}$ of $\mathcal{G}$ is an edge from $\mathcal{G}$. Two vertices in $\mathcal{L}$ are connected if and only if their corresponding edges (from $\mathcal{G}$) share a common endpoint in $\mathcal{G}$. Figure 4.2 demonstrates an example transformation with the key linegraph transformation occurring in the rightmost two figures.

By making use of the line-graph transformation on the starting graph $\mathcal{G}$, we can produce an attributed line graph $\mathcal{L}$ where each of the edges in $\mathcal{G}$ is represented by a node in $\mathcal{L}$. Rather than working with both $\mathcal{L}$ and $\mathcal{G}$, we create $\mathcal{G}'$ which combines aspects of both $\mathcal{G}$ and $\mathcal{L}$.

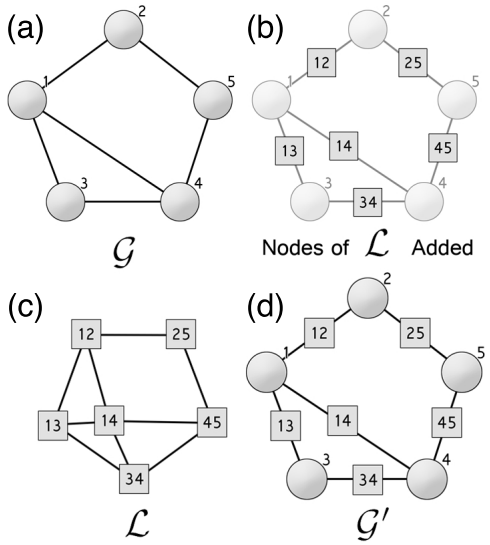


Figure 4.2: The linegraph transformation and the edge augmentation approach used to support edge and node attributes in MAGE. (a) shows the input graph. (b) is the intermediate step of the canonical linegraph transformation of $\mathcal{G}$, where a node for each original edge is created. (c) visualizes $\mathcal{L}(\mathcal{G})$ or $\mathcal{L}$ is the linegraph of $\mathcal{G}$ in which all edges from $\mathcal{G}$ that shared a node in $\mathcal{G}$ are now connected as the nodes of $\mathcal{L}$. (d) demonstrates edge augmentation wherein we embed the edge-nodes of $\mathcal{L}(\mathcal{G})$ directly into the original graph to create $\mathcal{G}'$.

We achieve this by transforming each edge in $\mathcal{G}$ into an edge-node in $\mathcal{G}'$. This new edge-node is connected to the same vertices in $\mathcal{G}'$ that it connected to as an edge of $\mathcal{G}$. This process is illustrated with a toy graph in Figure 4.2, where the edge-nodes are the square nodes splitting each edge of $\mathcal{G}$. Under this formulation, no two nodes in $\mathcal{G}$ will be directly connected in $\mathcal{G}'$. Similarly, no two newly introduced edge-nodes will be directly connected in $\mathcal{G}'$. The structure of the newly created graph is bipartite between the set of original nodes and the set of new edge-nodes. We make use of this fact in the development of MAGE.

4.3.1 MAGE Subroutines

We divide the process of discovering matching subgraphs into several key steps: (i) finding the initial nodes to start our localized search, then (ii) finding nearby nodes that fulfill or approximate the desired attributes and (iii) approximating the structure of the edges interconnecting nodes from steps i and ii.

```
1: procedure APPROXQUERY(G' A, Q')
2:     Initialization
3:     i ← Detect-Candidate
4:     i corresponds to a node u of Q'
5:     for all Nodes u in the query Q' do
6:         for all edges ⟨u, v⟩ from u to neighbor v do
7:             find node j by Local-Search
8:             approximate ⟨u, v⟩ with ⟨i, j⟩ via Linker
9:             edge ⟨u, v⟩ is fulfilled
10:        end for
11:    end for
12: end procedure
```

Figure 4.3: An overview of the approach used in finding an approximate subgraphs.

Detect-Candidate. The first node of each query approximation is located in the graph using Detect-Candidate in line 4 of Algorithm 4.3. This routine leverages attribute filtering and primarily subgraph centerpieces suggested in [172]. Detect-Candidate selects the most central node of the query first to help narrow the scope of the candidate search.

Local-Search. Given a partially fulfilled set of query nodes, Local-Search (line 8 of Algorithm 4.3) will locate a node in $\mathcal{G}'$ corresponding to an unfulfilled node from $\mathcal{Q}'$. It achieves this by using RWR values as weights for the nearby nodes with attribute matches. This approach will select *closer* to the approximated query, keeping the selected nodes in the local area of the current approximated nodes.

Linker. The Linker, line 9 in Algorithm 4.3, approximates the query's edges in the dictionary graph. If the two selected nodes of the real graph are connected the edge is returned; otherwise, the shortest path is used as an approximation of the edge. This shortest path may introduce additional nodes inorder to complete the pattern. The approximate solutions will add to the query, not remove from it.

4.3.2 Supporting Multiple Attributes

Often in real graph datasets, nodes and edges can have multiple fields of data. Similarly, a user may want to query for one or more attributes at the same time on a node or edge. These are important considerations and they are fully supported in MAGE by allowing lists of variables on each query node or edge. Each list of attributes are combined to produce query results with a logical OR for each item in the list during query time. This allows for the quick generation of queries across multiple node or edge attributes. Multi-attribute queries greatly extend the user's query possibilities and, therefore, exploratory power.

4.3.3 Supporting Wildcards

In order to ease the construction of query-graphs as well as extend usability of MAGE, we developed wildcard attributes. The wildcard allows MAGE to match any attributed node or edge to a wildcard node or edge, respectively, while maintaining the overall query connectivity. This is carried out by relaxing the node and edge attribute constraints used during the acquisition of nodes during query-result construction. Our experimental investigations suggest that our approach to wildcards is efficient and does not incur significant query overhead.

4.4 Methodology

The general approach taken for MAGE is explained in Procedure 4.4 and the subsequent procedures are described in the following sections.

4.4.1 Random Walk with Restart (RWR)

MAGE uses proximity scores between nodes in data graph $\mathcal{G}$ and query graph $\mathcal{Q}$ to determine how well a subgraph $\mathcal{Q}_i$ matches $\mathcal{Q}$. Specifically, the proximity between nodes i and j in a graph is the RWR value when node i is used as the restart point. The number of RWR

values needed to find a matching subgraph is significant, because we use them to rank possible candidate nodes. These calculations a major contributor to the overall runtime of the system and will be discussed in the experimental evaluation section. The RWR values in $\hat{F}'$ (see Equation 1) are used to rank the goodness of nearby candidate nodes; unselected nodes that are near to the partially constructed query receiver higher RWR values and are therefore more likely to be chosen. The power iteration multiplies a $\hat{F}$ by the sparse adjacency matrix $\mathcal{G}$ for the graph.

We have implemented a parallelized, sparse, power method that allows the fast calculation of RWR vectors. The canonical power-iteration method is a common approach to determine the RWR values see Equation (1). Decompose $\mathcal{G}$ row-wise into p submatrices each with m/p rows (call them $\mathcal{G}_1 ... \mathcal{G}_p$). Each $\mathcal{G}_i$ is zero everywhere except its m/p rows such that they sum to $\mathcal{G}$, see Equation (2).

$$\hat{F}_{k+1} = \alpha \mathcal{G} \cdot \hat{F}_k + \hat{Y} \tag{4.1}$$

$$\hat{F}_{k+1} = \alpha \left(\mathcal{G}_1 + \mathcal{G}_2 + ... + \mathcal{G}_p \right) \cdot \hat{F}_k + \hat{Y} \tag{4.2}$$

$$\hat{F}_{k+1} = \alpha \mathcal{G}_1 \cdot \hat{F} + \alpha \mathcal{G}_2 \cdot \hat{F}_k + ... + \alpha \mathcal{G}_p \cdot \hat{F}_k + \hat{Y} \tag{4.3}$$

For each node considered as a restart location the vector $\hat{Y}$ is set to the random restart probability $1 - \alpha$. Now using Equation (3) each submatrix-vector multiplication can be calculated separately in parallel and the results aggregated. Each iteration F_{k+1} is normalized to unit length and used in the next iteration. We choose only to synchronize the parallel computation at the end of each iteration rather than during it. This relaxation was chosen to maximize memory bandwidth during the calculation. We recommend choosing p as the number of available cores.

4.4.2 Primary Subsystems

Detect-Candidate and Local-Search. Both detecting the initial candidates and searching for nearby candidates follow the same general approach in our algorithm. At its heart the equation picks the new nodes heuristically based on their attribute and associated R score as calculated in Equation (4). The heuristic works by taking the product over all nodes adjacent to $v \in \mathcal{Q}'$ and summing their RWR scores with nodes $j \in \mathcal{G}'$ at least one of whose attributes matches a neighbor u's attribute.

$$R(\mathcal{Q}', i) = \prod_{u=Neigh(v)\in\mathcal{Q}'} f_j^{I(u,v)} \cdot \left(\frac{1}{n_u} \sum_{j_a \in u_a} f_j \right)^{1-I(u,v)} \tag{4.4}$$

Where n_u is the total number of nodes with attributes from u, is used as a normalizing factor. It's leveraged in order to keep common attributes from having over-sized scores. The sum goes over all nodes j with an attribute common with u The individual RWR values between nodes i and j are f_j calculated in Equation (1). Where $I(u, v)$ is the indicator function; returning 1 if edge (u, v) is already used in the partial query solution and 0 otherwise.

Linker. Given two nodes from $\mathcal{G}'$ the Linker computes a path connecting them. The implementation is an efficient modification of Prim's algorithm. We consider the "best path" as the path connecting the two nodes with largest RWR value when a direct link–one unused in the approximate solution–is not available. This score is the sum of RWR values over the whole length of the path.

4.4.3 Supporting Edge Attributes

In order to support edge attributes, a method is needed that allows the incorporation and later selection of data on each edge. For this we have chosen to embed an edge-node in each edge of $\mathcal{G}$, see Procedure 4.5.

Require: Fully-attributed graph $\mathcal{G}$, attribute graph $\mathcal{A}$, query graph $\mathcal{Q}$, and desired number of results k

Ensure: k node-attribute matched subgraphs from $\mathcal{G}$

procedure MAGE($\mathcal{G},\ \mathcal{A},\ \mathcal{Q},\ k$)
    $\mathcal{G}' =$ **Linegraph-Augmentor**($\mathcal{G}$)
    $\mathcal{Q}' =$ **Linegraph-Augmentor**($\mathcal{Q}$)
    $\mathcal{A}_w =$ **Wildcard-Attribute-Inserter**($\mathcal{A}$)
    for i=1:k **do**
        $\mathcal{Q}'_i =$ ApproxQuery($\mathcal{G}', \mathcal{A}_w, \mathcal{Q}'$)
        $\mathcal{Q}_i =$ **Linegraph-Reverter**($\mathcal{Q}'_i$)
    end for
return $\mathcal{Q}_i$ where $i = 1 : k$
end procedure

Figure 4.4: Detailed MAGE Algorithm

Linegraph Augmentation

The line graph augmentation algorithm is presented in Procedure 4.5 and operates in $O(m)$ where m is the number of edges from $\mathcal{G}$. This is precomputed a single time before querying begins.

Require: Edge-attributed $n \times n$ graph $\mathcal{G}$

Ensure: Edge-embedded $(m + n) \times (m + n)$ graph $\mathcal{G}'$ and a mapping M from edges to newly created edge-nodes

procedure LINEGRAPH-AUGMENTATION($\mathcal{G}$)
    Let $\mathcal{S}$ be an all-zero $(n \times m)$ matrix
    for all $u = 1 \rightarrow m$ edges, $e_{i,j} \in \mathcal{G}$ **do**
        $S(u, i) = 1$
        $S(u, j) = 1$
        $M(u) = e_{i,j}$
    end for
    $\mathcal{G}'$ **return** $\mathcal{G}'$
end procedure

Figure 4.5: Linegraph-Augmentation

This transformation creates $\mathcal{G}'$, a $(m + n) \times (m + n)$ adjacency matrix. Expanding both dimensions of our adjacency matrix by a factor of m may seem expensive in memory

usage; however, $\mathcal{G}'$ is guaranteed to be bipartite between the original nodes and the new edge-nodes. Because only original nodes can be connected to edge-nodes, we have only the $m \times n$ and $n \times m$ regions of our augmented matrix that can possibly contain values. We can derive the maximum matrix density, ρ_{max}, as follows:

$$\rho_{max} = \frac{2mn}{m^2 + 2mn + n^2} \tag{4.5}$$

if the graph is undirected only mn edges need to be stored. Because we use sparse data structures, the memory for this augmentation grows at a linear rate with the number of edges.

The matched subgraph results produced by MAGE are still embedded with edge-nodes and should therefore be converted back to the original graph format. The linegraph-reverter (see Procedure 4.6) serves the purpose of returning our modified results to the style and format specified with the input graph.

Require: Edge augmented query result $\mathcal{Q}'_i$
Ensure: Attribute matrix $\mathcal{Q}_i$

1: **procedure** LINEGRAPH-REVERTER($\mathcal{G}$)
2: **for all** node-edge $u_j \in Q'_i$ where $j = 1 \rightarrow q$ **do**
3: s = source edge leading into u_j
4: t = target edge leading out of u_j
5: remove edge $\langle s, u_j \rangle$
6: remove edge $\langle u_j, t \rangle$
7: replace edge-node u_j with edge from $M(u_j)$
8: **end for**
9: **return** $\mathcal{Q}_i$
10: **end procedure**

Figure 4.6: Linegraph-Reverter

4.4.4 Supporting Multiple Attributes

The categorical attribute matrix $\mathcal{A}$ is an $n \times t$ sparse matrix where there are t distinct attribute categories for each of n nodes. Each row of $\mathcal{A}$ represents a node while each

column represents a single categorical variable. Each node's categorical data is encoded as a–usually sparse–vector of ones.

While t can be very large, generally the mapping of categorical attributes to nodes is very sparse. Practically, $\mathcal{A}$ utilizes a minor amount of memory. We support multiple attributes on each edge and node, and allow them to be selected via logical OR. This is done by allowing the rows of $\mathcal{A}$ to have multiple values at once. These rows are leveraged during the attribute-centric RWR carried out in line 5 of Procedure 4.4. By serving as restart sources during the RWR calculations, the correctly attributed nodes are given larger proximity scores and therefore are more likely to be selected as a result.

4.4.5 Supporting Wildcards

To support the wildcard attribute we have created a universal attribute applied to all nodes and edges. This attribute is one among many that each edge or node may have at any time. The function labeled **Wildcard-Attribute-Inserter** on line 4 of Procedure 4.4 works by inserting a new and distinct attribute node in the attribute matrix A that points to all nodes. All that needs to be done to achieve this is to append a row of ones onto the attribute matrix $\mathcal{A}$. This technique works because the MAGE algorithm will select this wildcard attribute regardless of whatever other attributes a node or edge may have. When there are multiple choices to fulfill a wildcard the one with largest RWR value is selected, otherwise ties are broken by random selection.

4.5 Evaluation

We evaluate multiple important aspects of MAGE, such as speed, memory usage, and effectiveness on both large real and synthetic graph datasets, e.g., *460 million* edge Google+ graph[173, 174]. Experimenting on real graphs allows us to better understand how MAGE works in practice, while experimenting on synthetic ones let us carefully control experimental parameters and observe MAGE's responses. Besides investigating how MAGE's

speed scale with the graph size, we also study how supporting wildcards and multiple attributes in the graph queries would affect speed.

All tests were run on an Linux machine equipped with an Intel Core i5-4670K Haswell Quad-Core CPU (3.8 GHz) and 32GB of RAM. All code was written in C++.

4.5.1 Graph Datasets

Google+ Social Network. We have examined the scalability of our system using the attributed social network from Google+ [173, 174]. The dataset consists of four snapshots, each taken at different months of 2011. Their sizes range from 4.6M nodes and 47M edges for the smallest, to 28M nodes and 460M edges for the largest.

Rotten Tomatoes (RT) Movie Graph. We tested MAGE on a modest-sized directed graph constructed out of the Rotten Tomatoes (RT) movies. The graph contains more than 20,000 movies with edges connecting similar movies (based off of Rotten Tomatoes crowd-sourced film similarity). MAGE operates on categorical attributes, so continuous fields must be encoded categorically to be queryable in our system. For continuous fields, discretization offers several methods to aggregate multiple values into categories. We used quartiles to discretize the continuous movie-related values except for movie similarity which we discretized it to either weak or strong. In order to test the effectiveness of MAGE we queried this graph and present the examples results in Figure 4.11.

Synthetic graphs. For synthetic graphs we use stochastically generated Erdős-Rényi (ER) random graphs (parameterized by the number of nodes n and edges m) and Watts-Strogatz (WS) graphs (parameterized by the number of nodes n, the node degree k, and the rewiring probability p) [175, 176]. The parameter values used when generating the synthetic graphs are presented in Table 4.2. In the table, n and m are the number of nodes and edges, and k is the node degree used in the WS graph generation. The WS graphs are generated using $p = .01$ rewiring chance.

Table 4.2: Synthetic test graph sizes and parameters for each experiment. n and m are the number of nodes and edges, and k is the node degree used in the WS graph generation. The WS graphs are generated using $p = .01$ rewiring chance.

Experiment	Erdős-Rényi Graph (ER)	Watts-Strogatz Graph (WS)
Query (Fig 4.7)	$n = 1.5$M to 15M	$n = 10$M
	$m = 2$M to 162M	$k = 2$ to 10
RWR (Fig 4.8)	$n = 15$M	$n = 15$M
	$m = 15$M to 415M	$k = 2$ to 28
Wildcard (Fig 4.9)	$n = 10$M	$n = 2$M
	$m = 10$M	$k = 6$
Memory (Fig 4.10)	$n = 15$M	$n = 15$M
	$m = 15$M to 375M	$k = 4$ to 25

4.5.2 Scalability and Speed

RWR requires two parameters; the fly-out or restart-probability and the number of iterations, which we set to 0.15 and 10 respectively. MAGE performs queries in linear time with the number of edges in each graph (Figure 4.7). For both the ER graphs and the small-world WS graphs, the increase in query time is linear in the number of edges, suggesting good scalability. For the synthetic graphs attributes were randomly assigned to nodes and edge uniformly at random when the2 graph was generated. Three general query structures

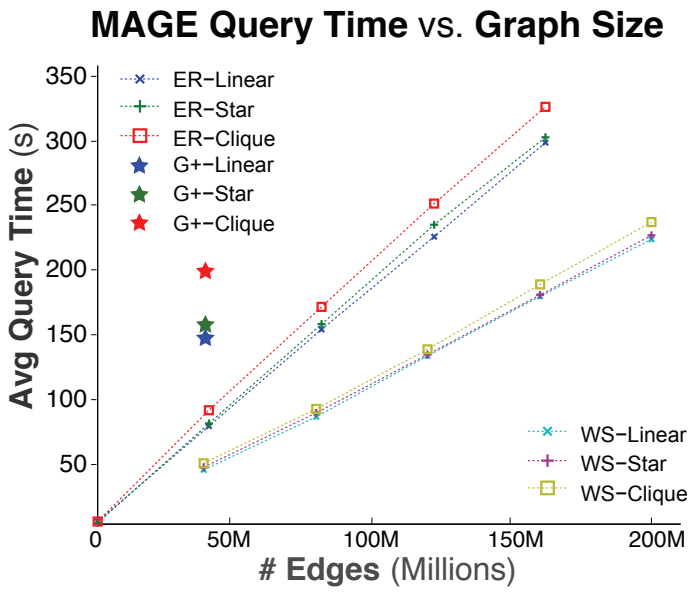


Figure 4.7: MAGE query time on real and synthetic graphs of varying sizes for a linear, star, and clique query. The average query time, over 10 runs, increases linearly with the (augmented) graph's size for each query type.

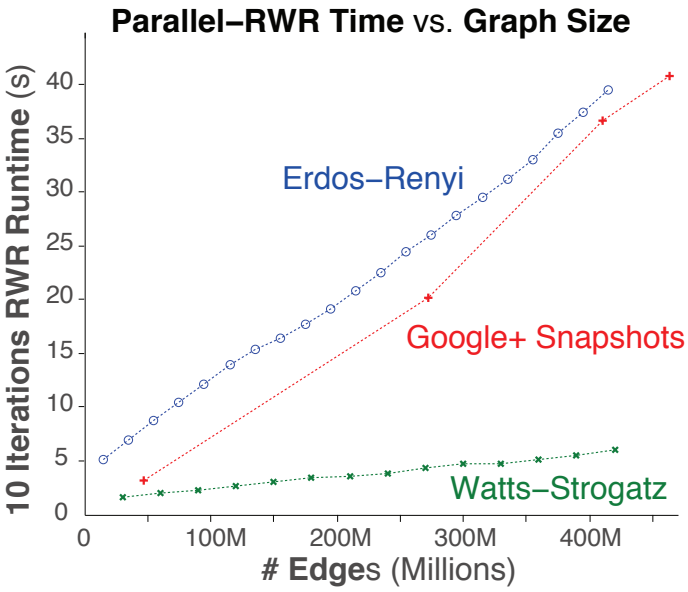


Figure 4.8: The runtimes for 10 iterations of the random walk with restart module over Erdős-Rényi, Watts-Strogatz, and Google+ graphs of varying sizes, which increase linearly with the number of edges, even into several hundred millions nodes and edges.

are explored for various sizes of graph; a 5-node 4-edge linear query, a 5-node 4-edge star, and a 5-node clique.

RWR Results.

The timely calculation of RWR values are essential to the MAGE algorithm. We test our approach on both synthetic WS and ER graphs as well as snapshots of various sizes from the Google+ social network dataset. The ER graphs were generated with randomly with increasing number of edges, while the WS graphs were generated with $(p = 0.01)$ and varying numbers of edges (see Table 4.2 for details). Each measurement is an average of the runtime for multiple stochastically generated networks at each m. The results for our parallelized RWR implementation are presented in Figure 4.8.

The ER graphs exhibit worse performance than WS graphs due to the constant random memory accesses during the parallelized matrix multiplication step. Our experimental results, both on real and synthetic graphs suggest excellent scalability of the core RWR algorithm.

Wildcard and Multi-attribute Cost.

In order to measure the potential added overhead of wildcards, we tested across multiple sizes of graph with varying numbers of wildcards in a common set of query. Multi-attributes are represented as multiple 1's in $\mathcal{A}$ and can be easily added for little overhead. Data-rich graphs can be queried in MAGE with minor increases in query latency and memory footprint. Multi-attributes make using wildcards far more powerful. For each number of wildcards, we averaged multiple runs and compare these against the same query graph with specified attributes. The queries used were a 6-node linear query, a 6-node star and a 5-clique. The wildcard counts in Figure 4.9 could were both edge and node wildcards selected at random.

Figure 4.9 shows the query time when using wildcards relative to a query with only normal attributes. The wildcard overhead increases linearly with the number of wildcards. This suggests that large queries can safely incorporate wildcards without computational

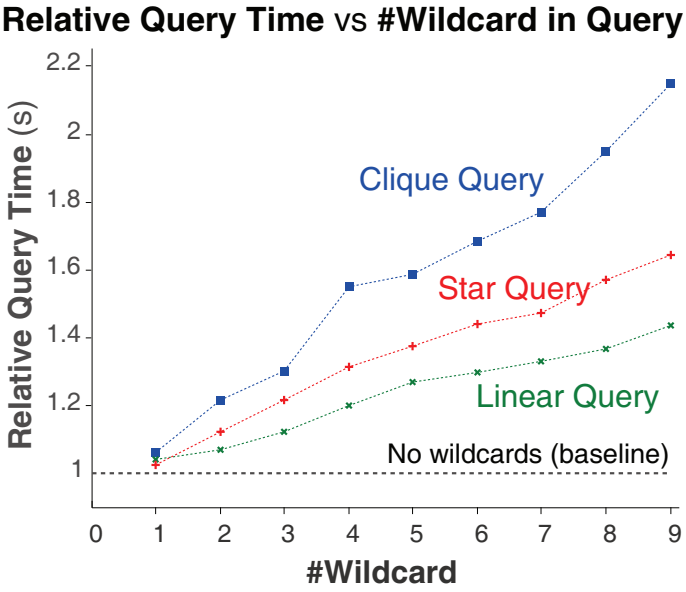


Figure 4.9: The relative response time for queries containing wildcards tested on an Erdős-Rényi graph with 10M nodes and edges, compared against the baseline query without wildcard (horizontal black dotted line). Query time is linear in the number of wildcards in the query.

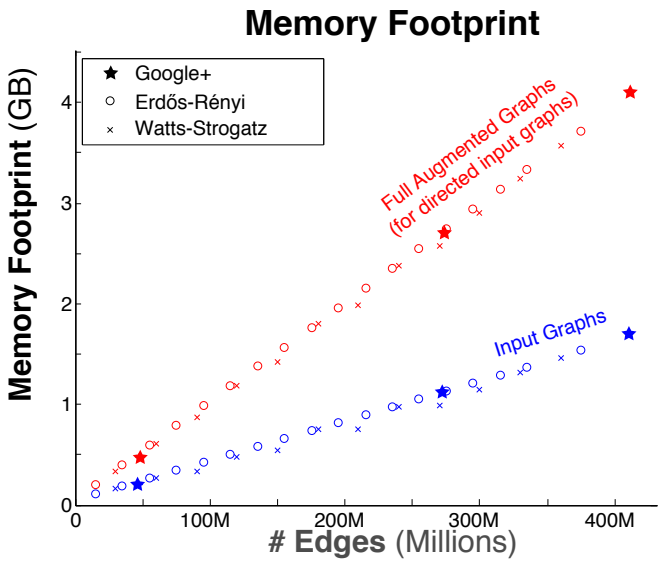


Figure 4.10: MAGE memory usage for Erdős-Rényi graphs and the Google+ graph, before and after edge augmentation, which increases linearly with the number of edges.

constraints, reducing the cognitive load on MAGE users during their data mining.

4.5.3 Memory Usage

We performed an experiment to measure the memory usage of edge-augmentation. In the directed case the full augmented matrix must be stored; however, in the undirected case only half of the matrix is necessary. We measured this cost because it is a potentially memory-intensive step. MAGE uses $2.5\times$ the augmentation memory during full querying. In Figure 4.10, we compare the memory footprint of each graph before and after edge augmentation (as previously explained in Secton 4.4.3) for graphs of increasing size.

4.5.4 Effectiveness

Measuring the effectiveness of approximate graph queries is a nontrivial task with two major challenges. The first problem stems from measuring similarity between the result and query, considering both structure and attributes. The second challenge is that the usefulness of a result is highly subjective. Human evaluation will be important in proving the overall effectiveness of our approach, and we plan to perform it in our future work. Here, we uses

existing work to inform the design of our experiments.

Method. Taking inspiration from related pattern matching work (e.g., [84]), we chose to modify graphs by injecting new patterns into them and then seeking those patterns with MAGE. With the approximations from MAGE, we can analyze the quality of matches with knowledge of the exact number of the desired pattern. We tested our approach against synthetic graphs and the RT movie graph. The synthetic graphs were generated in the same way as the previous experiments (see Section V-a); however, the patterns were generated with 6 distinct attributes per edge and 12 per node both of which were assigned randomly during graph generation. In the RT graph, attributes were randomly sampled from the empirical attribute distributions. In Table 4.3, we present 5 general query types: (i) a short line of 6 nodes, (ii) a long line of 15 nodes[1], (iii) a star with 15 spokes, (iv) a barbell with two 3-cliques connected by a path of length 3, and (v) a 7-clique [75].

For each query pattern, we calculate the average over the top 20 results for several

[1]Note that we also include a pattern that is a longer line, because short linear queries are trivially expressible in many other systems.

Table 4.3: The average similarity of the first 20 results and the query across several graph types. The λ similarity score is a modifcation of the Jaccard index and is the ratio of nodes and edges with attributes matched to the query normalized by the union of nodes and edges. Several types of queries, ranging in complexity, were tested over both sythetic (ER, WS) and real (RT) graphs. MAGE produces results with high similarity to the query.

Graph Type	Query Shape	% Extra Nodes	% Extra Edges	λ Score
Erdős-Rényi (ER)	Line (short)	11 $\pm$0.7	14 $\pm$0.9	0.89
	Line (long)	20 $\pm$0.8	26 $\pm$1.1	0.81
	Barbell	18 $\pm$0.7	18 $\pm$0.6	0.84
	Star	39 $\pm$0.8	44 $\pm$1.1	0.71
	Clique	64 $\pm$1.8	17 $\pm$0.1	0.77
Watts-Strogatz (WS)	Line (short)	0 $\pm$0.0	0 $\pm$0.0	1.00
	Line (long)	0 $\pm$0.0	1 $\pm$0.1	1.00
	Barbell	24 $\pm$3.5	28 $\pm$3.7	0.79
	Star	39 $\pm$6.5	51 $\pm$9.9	0.69
	Clique	104 $\pm$0.4	35 $\pm$0.0	0.66
Rotten Tomatoes (RT)	Line (short)	66 $\pm$1.8	30 $\pm$1.3	0.73
	Line (long)	84 $\pm$1.6	87 $\pm$2.2	0.54
	Barbell	96 $\pm$2.9	95 $\pm$1.9	0.51
	Star	104 $\pm$2.8	109 $\pm$2.6	0.50
	Clique	110 $\pm$3.4	103 $\pm$4.9	0.49

metrics. Table 4.3 displays the average %-difference in size between the queries and their matches. We also measure the quality of results using λ, a modified Jaccard Index measure over categorical variables which has been used in [177]. We compose the λ-similarity by taking the size of the set-intersection of nodes and edges from the query with the result and normalize it by the size set-union of the same query-result pair. When a result closely matches the query, $\lambda \rightarrow 1$, if the result has either extra nodes or nodes with different attributes then $\lambda < 1$.

MAGE's goal is to detect approximate matches to a user's desired query; Table 4.3 shows that the algorithm is capable of extracting results with good similarity. The WS graphs work well with linear queries, because their construction guarantees consistent paths, but because these graphs also have a high clustering coefficient, they respond poorer on clique queries. The RT queries generally perform worse than the synthetic graphs, because the RT contains several very low occurrence attributes. When included in a query, these rare attributes require an approximate path which adds additional nodes and edges, decreasing the λ-similarity. In general detecting exact and approximate cliques is a very challenging problem. When discovering initial candidate nodes for the results, MAGE may pick a structurally coherent (likely to be a clique) match, but with only some exactly matched attribute nodes. This means that some of the edges or nodes of the result will not be directly connected and will introduce some new nodes and edges. Even with the complexity of approximating 7-clique MAGE is still able to return good approximations, albeit with many extra nodes and edges, demonstrating MAGE's main strength is in finding approximate matches to help the user explore different hypotheses.

For the purposes of demonstrating the semantic result quality, we present visualized query results from the RT movie data in Figure 4.11. To demonstrate MAGEs generalizability, our effectiveness evaluation used multiple graphs, including a real Rotten Tomatoes movie similarity graph, where MAGE finds movies matching interesting criteria (e.g., finding a cult movie that has horror and comedy ingredients). MAGE was able to find the movie

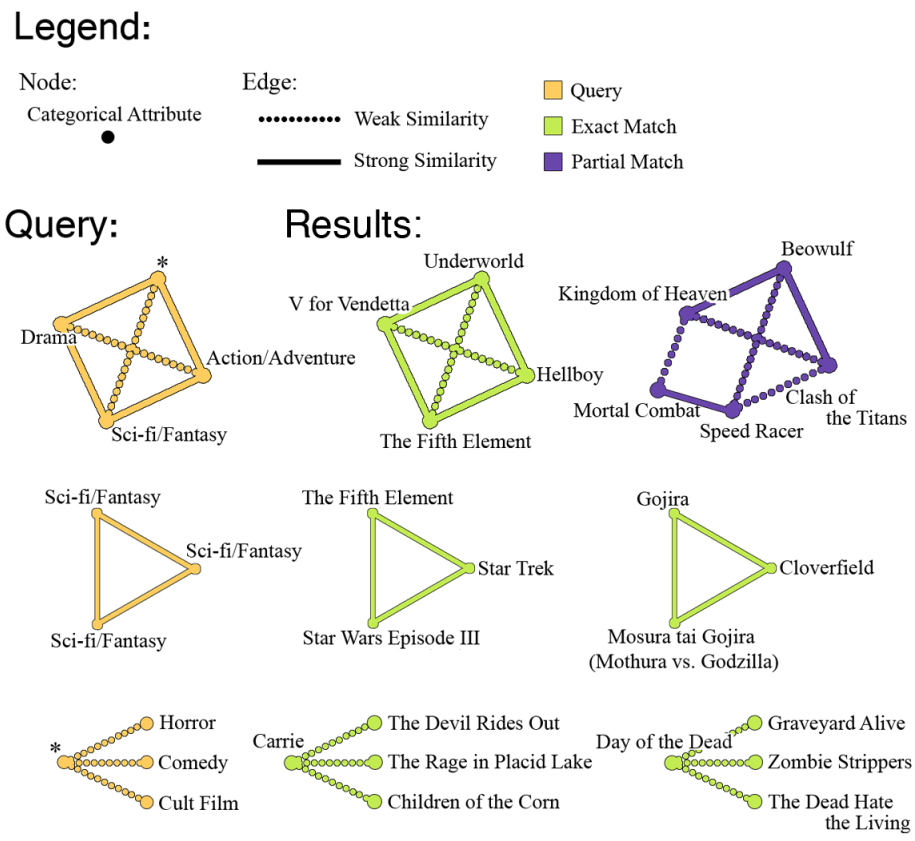


Figure 4.11: Rotten Tomatoes queries and example results. Each node represents a movie; each edge between two movies indicates that they are similar. The two edge classes (see legend) were derived from user-contributed similarity scores. Exactly matched results are presented in green while partially matched results are presented in purple.

Carrie that fits this query (see Fig 12, bottom row), which is mostly horror, with a few funny scenes; it indeed generated a cult following. Similarly, we also used MAGE to find movies that occupy the intersect of the genres of drama, sci-fi and action (as seen in Fig 12, top row) and it returned two popular films *Underworld* and *Beowolf* that fit this description very well. This experiment with the RT graph suggests that MAGE has the potential to be used in consumer-facing, main-stream applications such as movie recommendation.

4.6 Conclusion

To the best of our knowledge, MAGE is the first approach that supports exact and approximate subgraph matching on graphs with both node and edge attributes, for which wildcards

and multiple attribute values are permissible. Our experiments on large real and synthetic graphs (e.g., *460M* edge Google+ graph) demonstrate that MAGE is both effective and scalable. Using multiple node or edge attributes in a query incurs negligible costs, and MAGE scales linearly with the number of wildcards. To demonstrate MAGE's generalizability, our effectiveness evaluation used multiple graphs, including a real Rotten Tomatoes movie similarity graph, where MAGE finds movies matching interesting criteria (e.g., finding a cult movie that has horror and comedy ingredients). We believe MAGE can be a helpful tool for exploring large, real-world graphs.

CHAPTER 5

VISAGE: INTERACTIVE VISUAL GRAPH QUERYING

Building on our research from the previous section, we constructed a visual, interactive graph querying system. We present VISAGE, an approach that empowers users to construct expressive queries, without writing complex code.

5.1 Introduction

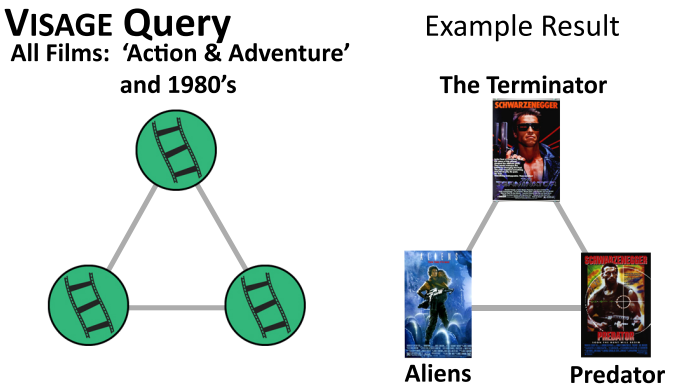
From e-commerce to computer security, graphs (or networks) are commonly used for capturing relationships among entities (e.g., who-buys-what on Amazon, who-called-whom networks, etc.). Finding interesting, suspicious, or malicious patterns in such graphs has been the core enabling technologies for solving many important problems, such as flagging "near cliques" formed among company insiders who carefully timed their financial transactions [178], or discovering "near-bipartite cores" formed among fraudsters and their accomplices in online auction sites [84]. Such pattern-finding process is formally called *graph querying* (or *subgraph matching*) [75, 79].

Many graph databases now support pattern matching and overcome the prohibitive costs of joining tables in relational databases [182]. Specifying graph patterns, unfortunately, can be a challenging task. Users often need to overcome steep learning curves to learn querying languages specific to the graph databases storing the graphs.

For example, many graph databases store graphs in the Resource Description Framework (RDF) format, which capture subject-predicate-object relationships among objects[1]. These systems support the SPARQL querying language, which is hard to learn and use

Chapter adapted from work at AVI'16 [Paper Link] [179]
Demo of the work received Best Demo Honorable Mention at SIGMOD'17 [Paper Link] [180]
Poster version of the work at IUI'15 [Paper Link] [181]

[1] `http://www.w3.org/RDF/`



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 5.1: *Top*: a VISAGE query seeking three similar action films from the 1980's along with a result, found from the RottenTomatoes movie-similarity graph (an edge connects two movies if they are similar). *Bottom*: the equivalent query written in the Cypher querying language. VISAGE's interactive graph querying approach significantly simplifies the query writing process.

[183]. The Cypher language, designed for the recent Neo4j graph database[2], is easier to work with since its syntax more closely resembles SQL [184, 185], but expressing relationships among nodes can still be challenging and may require writing many lines of code even for conceptually simple queries [186], as demonstrated in Figure 5.1, which seeks a "triangle" of three similar action films from the 1980's [76].

While there has been a lot of work in developing querying algorithms (e.g., [75, 79, 76]), there has been far less research on understanding and tackling the visualization, interaction, and usability challenges in the pattern specification process. Studying the user-facing aspects of subgraph matching is critical to fostering insights from interactive exploration and analysis. While early works suggested such potential [77, 82, 86], none

[2] http://neo4j.com/

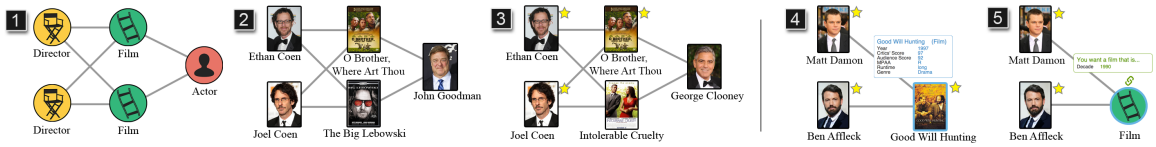


Figure 5.2: VISAGE supports many query refinement approaches like abstract querying (1-3) and example-driven querying (4-5). A broad query (1) with only node types and structure, with the first resulting match in (2). The Coen Brothers and the film *O Brother, Where Art Thou?* are starred, fixing these nodes. With the nodes starred, only matches with those nodes are displayed like (3). Bottom-up querying or query by example starts with an example of a known pattern. The known pattern (4) coveys lots of detailed information but is too specific to offer any other matches. In (5), *Good Will Hunting* is abstracted to form a new query based off the example (for only films from the 90s).

evaluated their ideas with users. Hence, their usability and impact are not known.

We propose VISAGE, the **Vis**ual **A**daptive **G**raph **E**ngine[3], which provides an adaptive, visual approach to graph query construction and refinement, to simplify and speed up graph query construction (Figure 5.3). 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 an interaction technique for graphs called *graph-autocomplete* that guides users to construct and refine queries as they add nodes, edges, and conditions (feature constraints). Adding too many nodes, edges, or conditions may result in over-specification (too few results) or even a null-result (no results found) [60]. Graph-autocomplete stops the user from constructing null-result-queries and guides the query-specification process.

- We design and develop a 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 [77, 82]. In the abstract case, users start with a very abstract query and narrow down the possible results by providing feature and topological constraints. In the example-driven case, often called *query by example* (QBE) [58], users can specify an exact pattern and abstract

[3]Please see video-demo in supplementary material.

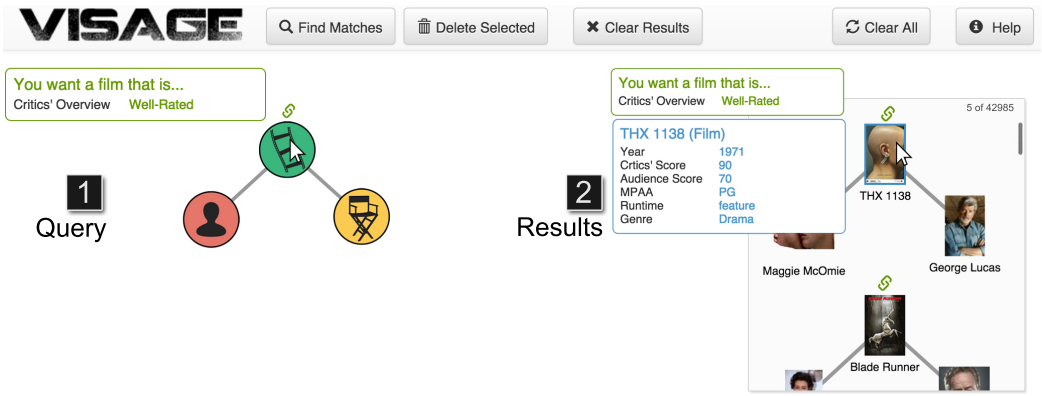


Figure 5.3: A screenshot of VISAGE showing an example graph query of films and actors related to George Lucas' films. VISAGE consists of: (1) a *query construction area*, where users construct graph queries by placing nodes and edges; (2) an *overview popup window* that summarizes the desired features (constraints or conditions) of a query node (in green), and the features of a selected node in a match (e.g., the film *THX 1138* in blue); a *results pane*, which shows a list of the results returned by the query. In this example, a user has specified a condition that the film must have a critics' overview of "Well-rated". The matches' layouts (general shape) mirror that of the original query.

from that pattern into a query of their choice. This technique allows users to start from an example or keep a value fixed in their query. In VISAGE, the user 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 demonstrate VISAGE's ease of use and the ability to construct graph queries significantly faster than conventional query languages, through a twelve-participant, within-subject user study.

5.2 Scenario

We provide two scenarios to illustrate how users may use VISAGE. The first scenario starts from a general question with a known structure and narrows the search through query refinement. The second scenario begins with a known example from which new similar results are found through abstraction.

The Rotten Tomatoes Movie Graph Throughout this chapter, we use a Rotten Tomatoes[4] film-actor-director graph. 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.

Scenario 1: From Abstract to Detailed Imagine our user Lana wants to find co-directors who have starred the same actor in two films. She can begin specifying her query starting with very general terms. She right clicks the background and chooses to add a new director node, she repeats this to add another director, and again to add two films and an actor. She attaches the director to the films and the films to the actor (see Figure 5.2.1), by clicking and dragging from one node to the other (one pair at a time). She clicks the search button. She gets the results in the results list, we show only the first result (in Figure 5.3.2) to save space. She likes the first result (in Figure 5.2.2) 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 the query. She performs the search again with these values fixed. 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 5.2.3, showing George Clooney in *Intolerable Cruelty*.

Scenario 2: Building up From a Known Example Now consider the example-driven approach, where a user, Barry, takes a known example and abstracts it into a new query. Barry knows that Matt Damon and Ben Affleck both starred in *Good Will Hunting*, so he draws a node for each person and one for the film, and connects each actor to the film (see Figure 5.2.4). Specific nodes can be added manually by searching for them in the node search menu (see Figure 5.4.1). When a specific node is added via search, it's automatically

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

starred so that its value will remain fixed in the query. Nodes can be unstarred by clicking the star icon in the upper right corner (see the star by Matt Damon in Figure 5.2.4). Because Barry starred all the nodes in his query, searching only finds one result (if Barry was wrong and his initial example does not exist, no results will be shown and he will be alerted via text in the empty results panel). By specifying the exact value of the nodes, the query has become too specific and will need to be abstracted if Barry wants more results. Barry then *unstars* the specific film *Good Will Hunting*, to find any movie starring both actors (Figure 5.2.5). Barry can also leverage the visualized features of *Good Will Hunting* to specify new constraints based on the results (i.e., only selecting movies made in the 1990's co-starring the actor duo). He uses a visualization of the possible constraints discussed in Section 5.4.1 and shown in Figure 5.5. Barry reissues his search and finds *Dogma* (among others), a potentially exciting film for him to watch.

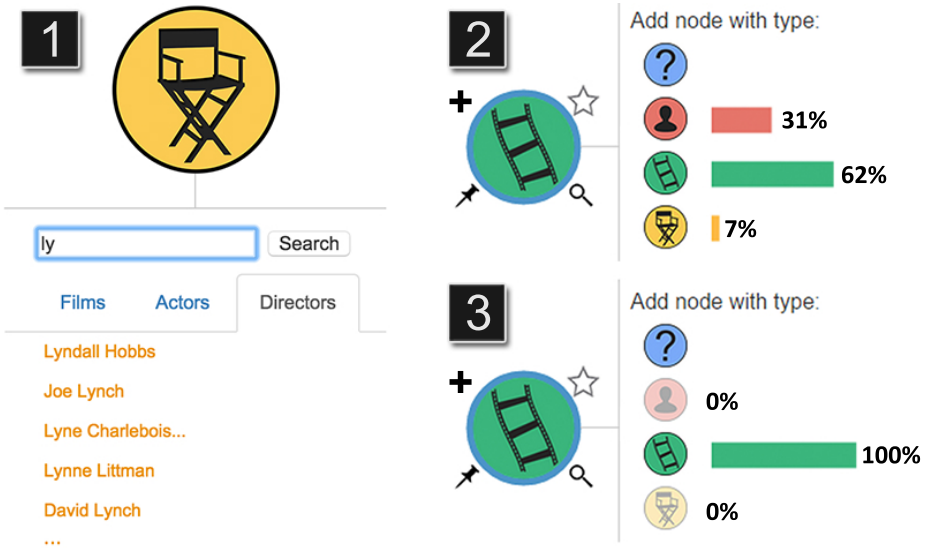


Figure 5.4: VISAGE offers several features to ease the selection of individual node values. (1) VISAGE supports conventional text search for finding a node to star. (2) Node controls and the add-node menu; the pin button fixes the node's position in the visualization; the star button (available only when results exist) allows users to keep that particular node in future results; the magnifying glass opens the node-search menu (at 1) that allows users to search for particular nodes. (3) The distribution of each potential neighbor node type is plotted to the right of each node-button; neighbors that will lead to over-specification are grayed out.

5.3 VISAGE Overview

The user interface for VISAGE is comprised of a force directed query-graph visualization (Figure 5.3), a context menu that provides an overview of features (Figure 5.3.2 in blue), a feature exploration pane (Figure 5.5), and a results list (Figure 5.3.2). The graph view shows the current state of the user's query. Matches are found in the background during interaction with the feature tree and query construction, but can be fetched manually using the "Find Matches" button at the top. Results are displayed in a popup list (Figure 5.3.3) which can be removed by clicking "Clear Results" at the top.

When users select a node, a blue border appears along with the node context menu (Figure 5.3.2). The context menu shows the current selected feature constraints in green (if the user wants the selected movie to only have good ratings then they can select this constraint in the feature tree in Figure 5.3.3). 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 (Matt Damon in Figure 5.2.4) 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 5.4.2). This menu displays the types of nodes that, if added, guarantee at least one match in the underlying network. 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 star the node, keeping it constant in the query. The magnifying glass opens the node search menu, in Figure 5.4.1, 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 VISAGE allows the user to form complex graph queries, where the nodes can be as abstract as a wildcard or as constrained as taking a single value

(recall the scenario with the Coen brothers in Figure 5.2). Graphs with a known type, for example a film, can have any number of additional constraints added to them; limiting the possible matches in the underlying dataset. The feature-tree allows users to explore hierarchical and non-hierarchical features (for both categorical and continuous variables).

5.4 Design Rationale

Supporting Expressive Querying: Abstract to Specific Graph querying requires the user to specify a group of nodes and their relationships; however, the constraints placed on the nodes can range wildly, from specific to abstract (e.g., a wildcard node of any type). A key design goal was to allow users to express their queries ranging from abstract to very specific. Users may start from known examples and abstract based off of the features of their example.

We are able to leverage the internal capabilities of Neo4j in terms of query conditions and indexing to support more complex queries. We support true wildcards (which can take on any node type and value). We use indices to support constant-time lookup for all starred nodes. Conditions are added by clicking on that value in the feature hierarchy. Users are free to add as many as they like. Within each feature (whether flat or hierarchical), we employ the logical *or* operation for constraints (i.e., year = 1997 **or** year = 1998). Across the features, we use a logical *and* for the constraints (i.e., genre = horror **and** year = 1988).

5.4.1 Improving Visual Query Refinement: Autocomplete

Graph autocomplete has two primary goals (1) keeping the user from making queries with no results and (2) helping them understand the features of the matches of their query during refinement.

Structural Guidance When a user submits an over-specified query (one that has too few or no matches), they must return to their query and refine it until they reach a suitable level

of specification. To help avoid this, we adapt the query construction process based on the query the user is constructing. VISAGE directs the user's query construction towards results by providing critical information about possible nodes and their features. We have created the first graph-querying *autocomplete*, which works on **node types**. We want to limit the types a nodes users can add so that their query always has at least one result. This guides the user in the direction of queries that are rich in matches and away from over-specification and null-results [60].

We achieve type- or structure-autocomplete by constraining the possible-neighbor options in the new node menu. By querying in the background we determine which types of newly added nodes and edges will result in matches and which won't. VISAGE displays this data by desaturating the add node button. This way a user can immediately see which types of nodes are available to them. In the case of truly massive graphs, background querying for node-types may be too slow. In this case, we use first-k-sampling to guide the user. We use the first k-results of the current query to determine the feasibility and distribution of potential new nodes given the current query. The samples are visualized in the bar graph to the right of the node-buttons (Figure 5.4.1).

Feature Guidance Graph-autocomplete also works in the feature space. We do this by visualizing the distributions of different node-attributes, from a sample of the results, of the current query. This approach provides users with detailed information about how the features (of their current queries) are distributed. With knowledge of different attributes, users are able to better understand how the results fill out the feature space. These data provide a visual cue that indicates how a new condition will change the number of results.

We chose to visually encode the feature frequencies in the edges of the feature-tree with edge-width and saturation (Figure 5.5.2). By adding constraints with sparser features (thin, light lines in Figure 5.5.2), users will quickly decrease the number of matches. If users choose denser attributes they will constrain their search less, keeping more of the results.

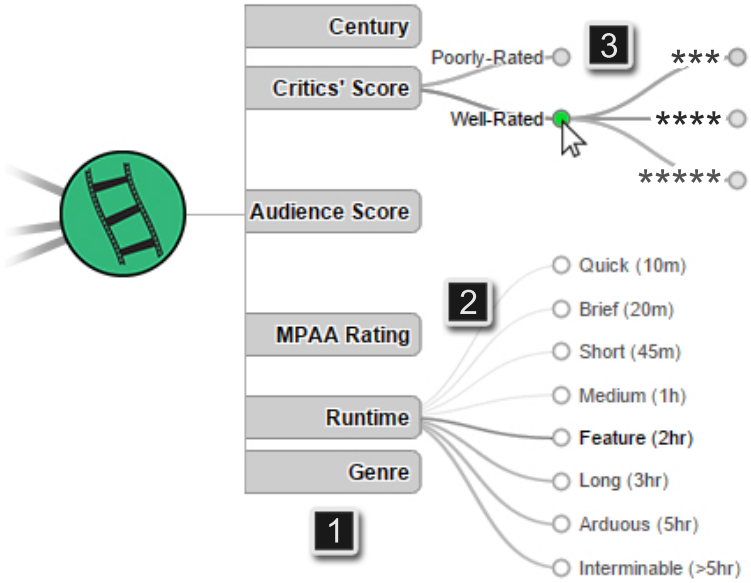


Figure 5.5: To investigate the feature space, VISAGE visualizes node features with a tree view. Hierarchical node features can be clicked to hide or show levels of the hierarchy (3). The edges denote the density of the target feature in the current results. The darker edges in (2) mean more results have that attribute value. When a user adds a condition by clicking a node it is highlighted in green, as in (3). If the current node is a result or a starred node, that nodes attributes will be highlighted in blue.

The feature tree also promotes abstraction in hierarchical attributes (Figure 5.5.3), because it is straightforward to trace from one constraint up to the parent constraint. For example, instead of looking only for films from 1993, the user can move up the hierarchy seeking films from the 1990's. Feature-autocomplete gives users the summary feature information needed to narrow their search without having to repeatedly go back and forth from query to results.

5.5 Implementation

VISAGE uses a client-server architecture (Figure 5.6) that separates the front-end interactive visualization (client) from the backend graph matching and storage (server). We have designed VISAGE to be independent of its backing graph database. VISAGE fully supports Neo4j [7]. Currently, it also partially supports SPARQL, with full support in the near future. VISAGE's web client (Javascript and D3) and server (Python) can run smoothly on the

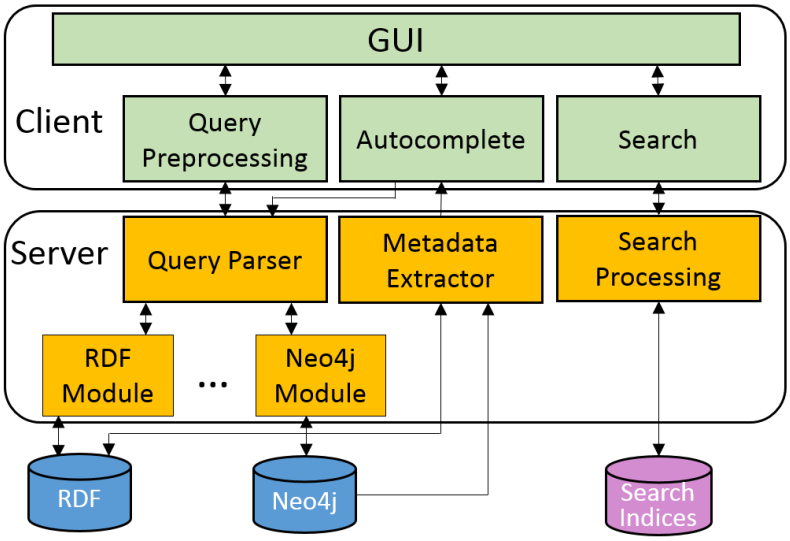


Figure 5.6: VISAGE uses a client-server architecture. The client visualizes the query and results. The server wraps a graph database, e.g., Neo4j, RDF database; additional databases can be added via new parser output modules. The *metadata extractor* creates summarization statistics for autocomplete. VISAGE's search functionality for finding specific nodes is sped up using full-text-search indices.

same commodity computer (e.g., we developed VISAGE on a machine with Intel i5-4670K 3.65GHz CPU and 32GB RAM). Optionally, for larger graphs, the server may be run on a separate, more powerful machine.

To fetch results of a user's query, we convert and parse the visual query into a compact format that we pass off to the DB modules which convert the parsed query into the necessary languages for each graph database. Once results are returned, we calculate summary statistics with the metadata extractor in Figure 5.6, which are the input for graph-autocomplete and represent the results of the current query.

Parsing A Graph Query When looking for matches of a query, if the starting node has very few matches in the graph, the search space is reduced and fewer comparisons are needed. The effect can be enormous, reducing a multiple minute query down to sub-second times. Because the node-constraints can vary from completely abstract (like a wildcard) to a single specific node, we have designed VISAGE to partition the graph queries into pieces. Our first step is to rank the nodes by the number and severity of their constraints. Starred

nodes are parsed into subqueries first. VISAGE then ranks the remaining nodes by number of constraints. The entire parsing can often be completed in a few milliseconds or less.

5.6 User Study

To evaluate VISAGE's usability, we conducted a laboratory study to assess how well participants could use VISAGE to construct queries on the Rotten Tomatoes movie graph discussed earlier. We chose a movie graph, because the concept of films, directors, etc., would be familiar to all participants, so that they could focus on VISAGE's features. We asked participants to build queries to find interesting graph patterns derived from prior graph mining research [92, 76, 93]. We compared the time taken forming queries between VISAGE and Cypher; we chose Cypher for its resemblance to SQL and ease of use. We chose Participants were not informed which system, if either, was developed by the examiner. We are not able to compare with GRAPHITE [77], as it is not publicly available.

Participants

We recruited 12 participants from our institution through advertisements posted to local mailing lists. Their ages ranged from 21 to 31, with an average age of 25. 7 participants were female the rest were male. All participants were screened for their familiarity with SQL. Participants ranged in querying skills; three had prior experience with Cypher. Each study lasted for about 60 minutes, and the participants were paid $10 for their time.

Experiment Design

Our study uses a within-subjects design with two main conditions for completing tasks: **VISAGE** and **Cypher**. The test consisted query tasks which were divided into two sections (see the *Task* section). Every participant completed the first section of tasks in one condition, and the second set of tasks in the remaining condition. The order of the conditions was counterbalanced. We generated matched sets of tasks, Set A and B, each with 5 tasks

to complete. The tasks ranged from easy to hard and were counterbalanced with each condition, to even out unintended differences in difficulty among the tasks. We used two sets of tasks to ensure that participants did not remember their solutions between sets.

Tasks

We created the tasks based on an informal survey of interesting patterns from common questions people formed when exploring Rotten Tomatoes data and from prior graph mining research [92, 76, 93]. The tasks in Task Set A (shown in Figure 5.7) were:

1. Find films similar to any film from *1993*.
2. Find an actor and a director for any *drama* film.
3. Find an actor starred in 3 films: *romance*, *comedy*, and *action*.
4. Find 3 similar *action* films, where one is from the 80's, one from the 90's, and the last from the 00's.
5. Find co-directors who made at least three films together, starring the same actor, where one of the films was *from the 90's*.

The difficulty of each query increases from task 1 through 5. We ranked the difficulty of each task based on the amount of Cypher code and number of nodes, edges, and constraints needed. The italic values shown above were the elements that differed between the two task sets (the order remained the same across both sets). Our hypothesis was that Cypher and VISAGE would achieve approximately similar performance for easy queries and VISAGE would achieve shorter task completion times for harder queries.

Task completion time was our dependent measure. Task completion time could be affected by: (1) Software – VISAGE or Cypher; (2) Task Set – the Task Set A or B; (3) Software Order – which software was used first. Using a Latin square design, we created 4 participant groups (since all subjects would do both software systems). We randomly and evenly assigned the participants to the groups, e.g., one group is *(*VISAGE + *Task Set A)*

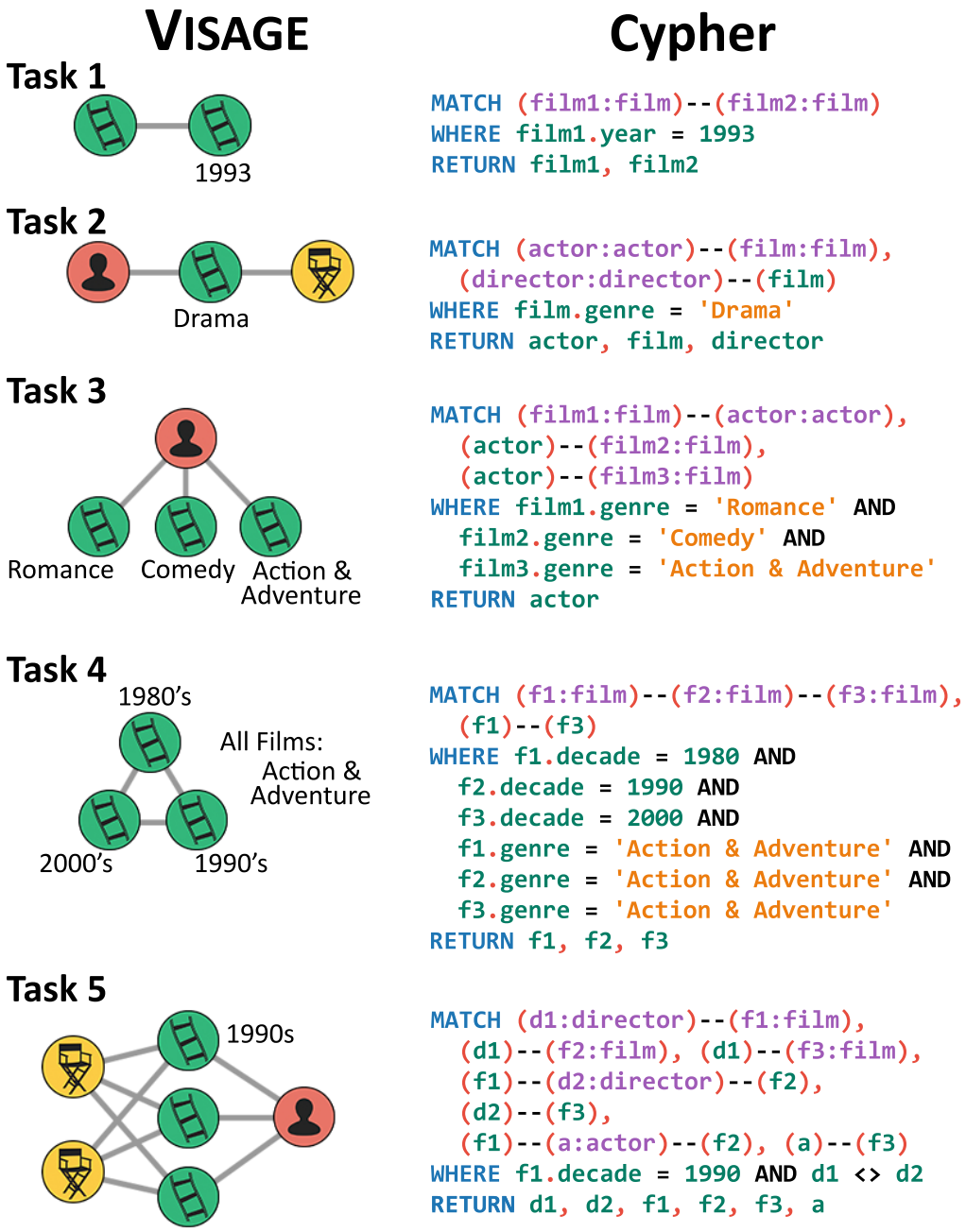


Figure 5.7: VISAGE user study tasks. VISAGE queries shown on the left with their corresponding multi-line Cypher queries on the right.

then (Cyper + Task Set B).

Procedure

Before the participants were given the tasks, they were provided with instructions on the software that they would be using as well as information about the data set they would be exploring. For the Cypher querying language, we offered a tutorial for starting Cypher tailored to our dataset. For VISAGE we provided an overview of VISAGE's interface, how to construct queries, and how our tool would work. The participants were welcome to ask clarifying questions during these introductory periods.

Once demoed we moved on the the first block of tasks, where we instructed the participants to work quickly and accurately. They had 5 minutes to perform each task and could only move to the next one if they correctly completed the current task or ran out of time. After each task, the participant was given the next task's instruction while the system was reset. Each task was timed separately. If a participant failed to finish a task within the allotted 5 minutes (300 seconds), the experimenter stopped the participant, marked that task as a failure, and recorded 300s as the task completion time (to prevent participants from spending indefinite amounts of time on tasks).

Once participants had completed the first set of tasks, they were provided the next set. At the end participants completed a questionnaire that asked for subjective impressions about each software system.

5.6.1 Results

Quantitative Results

The task completion times were analyzed using a mixed-model analysis of variance with fixed effects for *software, software order, task set, and a random effect across participants.* This technique is used to analyze within-subject studies and improves over conventional ANOVA, because error-terms are also calculated per-subject [187].

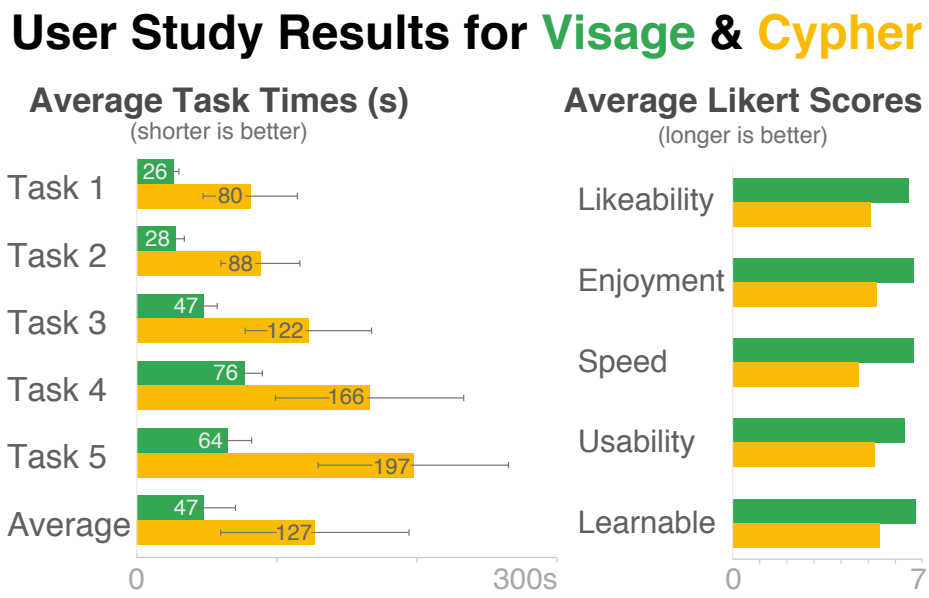


Figure 5.8: Average task completion times and likert scores for VISAGE (green) and Cypher (yellow). VISAGE is statistically significantly faster across all tasks. The error bars represent one standard deviation.

We measured the task completion time for the effects over all possible combinations of software, software order, and task set. The only statistically significant effect was software, suggesting a successful counterbalancing of software order and the equality of the difficulty of the two task sets. Figure 5.8-left demonstrates the average time per task for the study. The software effect was significant across all tasks: task 1 ($F_{1,5} = 27.16,\ p < 0.0004$), task 2 ($F_{1,5} = 49.76,\ p < 0.0001$), task 3 ($F_{1,5} = 33.23,\ p < 0.0002$), task 4 ($F_{1,5} = 25.88,\ p < 0.0005$), task 5 ($F_{1,5} = 42.84,\ p < 0.0002$). Participants were significantly faster when constructing queries in VISAGE than in Cypher. Only one participant failed to complete task 5 in the allotted 5 minutes (using the Cypher software); the rest succeeded in all tasks. This datum is partially responsible for the high variance in the task 5 (see Figure 5.8-left - Task 5). Using VISAGE, participants were able to construct task 5 slightly faster than task 4 (see Figure 5.8-left). Adding new nodes in VISAGE is faster than specifying feature constraints; task 4 has a large number of constraints while task 5 has a large number of nodes and edges. We do not see this in Cypher task 4 and 5, because adding new edges, nodes, and constraints all take similar amounts of time. Overall, the average difference in

task times between VISAGE and Cypher was statistically significant ($F_{1,5} = 37.38,\ p < 0.0005$); this represents an average speedup of about 2.67× when using VISAGE.

Subjective Results

We measured several aspects of both conditions using 7-point Likert scales filled out at the end of the study. Participants felt that VISAGE was better than Cypher for all the aspects asked about (see Figure 5.8-right). The participants enjoyed using VISAGE more than a querying language and additionally found that our system was easier to learn, easier to use and more likeable overall; although this is a common experimental effect, we find the results encouraging. Several participants found that the visualization of the query greatly improved the overall completion of the tasks.

5.6.2 Discussion and Limitations

The results of our user study were positive, both qualitatively and quantitatively. This suggests that VISAGE's visual representation of graph queries using graph autocomplete is faster than typed querying languages. We believe that VISAGE achieves these better times by: (1) streamlining the process of adding nodes and edges; (2) autocompleting partially-complete queries, which adaptively guides the user away from null-results; (3) shielding users from making typos and mistakes during the construction of their queries.

Adding nodes and edges in traditional querying systems often requires creating a variable for them, which must be remembered in order to specify the structure and attributes related to it. The user may have to type the name a single node repeatedly in order to specify the actual structure and in the case of large queries may confuse the names of nodes. VISAGE's visual representation simplifies this considerably. Typos and mistakes are common when writing a long and complicated query by hand. By programmatically generating queries based on users' constructions, VISAGE avoids the delay incurred by typos.

We observed two general strategies that participants employed when constructing

queries: (1) entities first, then relationships, and (2) iterative construction. Participants in the first group would often add all the entities from the task first, then wire up the relationships. Other participants followed a more iterative approach, wherein they would start with a single entity and build up from it (reminiscent of a breadth-first search). No statistical significance was found in the time taken for each general strategy.

When users construct queries with null-results, a traditional system requires the user to wait while the search is performed. During the study two of the users stated that the autocomplete helped remind them about the underlying structure of the network, saving time during their tasks. We help guide the user away from this case with our graph-autocomplete, so that users spend less time debugging queries that do not produce results. Because we sample results for graph-autocomplete, VISAGE may be able to retrieve a small sample of the possible results in real time, leading to a potentially skewed samples. This limit is dictated by the underlying graph database and scales accordingly.

While the results of our evaluation was positive, the need for the participants to build queries was created by the tasks; however, in real-world scenarios, such needs would be ad hoc. For example, what kind of exploratory query patterns do people create? We plan to study such needs, as well as how VISAGE can handle those kinds of tasks, in less controlled situations.

5.7 Scalability and Query Latency

We performed a series of tests to investigate the speed of our system. We tested VISAGE in two ways: (1) using basic queries and scaling their size and (2) by using examples that study participants looked at when given free use of the system.

Scalability Experiment 1: Varied Query Size

For the first analysis, we selected three candidate queries of various sizes (see star/claw, clique and line from Figure 5.9) and run them 10 times in VISAGE, a random feature

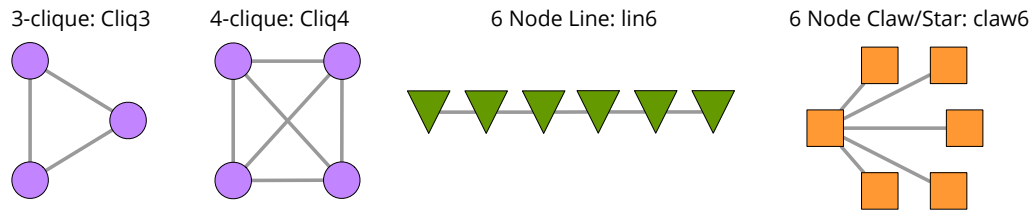


Figure 5.9: Examples of some the queries used in the query scalability tests. Colors and node shapes are matched to the style in Figure 5.10 where the run times for various sizes of query are shown.

condition was added to each node. The average runtimes for each size of query are reported in Figure 5.10-left. Relying heavily on indices, VISAGE operates fast enough to provide real-time results.

Because VISAGE is separate from the underlying graph querying database, we tested the speed with which we can parse a visual graph result into a Cypher Query. In Figure 5.10-right, we show the amount of time to parse versus the number of nodes in the query. VISAGE parses the query quickly, such that the graph database search time is far larger.

Scalability Experiment 2: Common Patterns with Constraints

The second test used common query patterns with numerous feature constraints, visualized in Figure 5.11. The runtime performance for each of these queries is shown in Figure 5.12.

The first query, *Timeless Barbell*, illustrates a query for an actor who was in two similar movies from the 2000's and co-starred with an actor in two similar films in the 1970's.

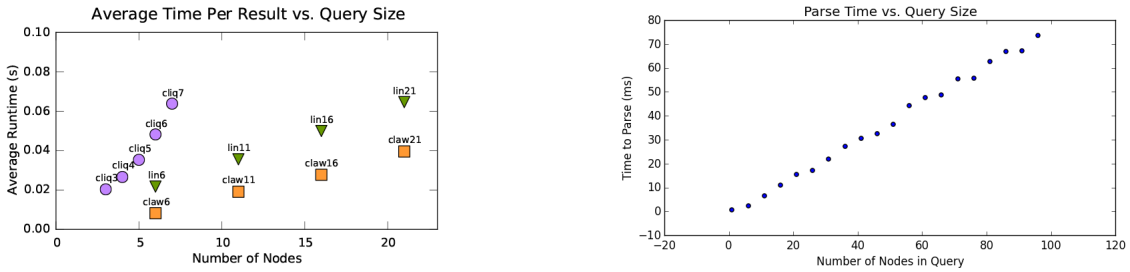


Figure 5.10: Left: Run times for varied sizes of queries in VISAGE. Right: time taken to parse visual input into a Cypher query. VISAGE operates quickly, even for complex queries.

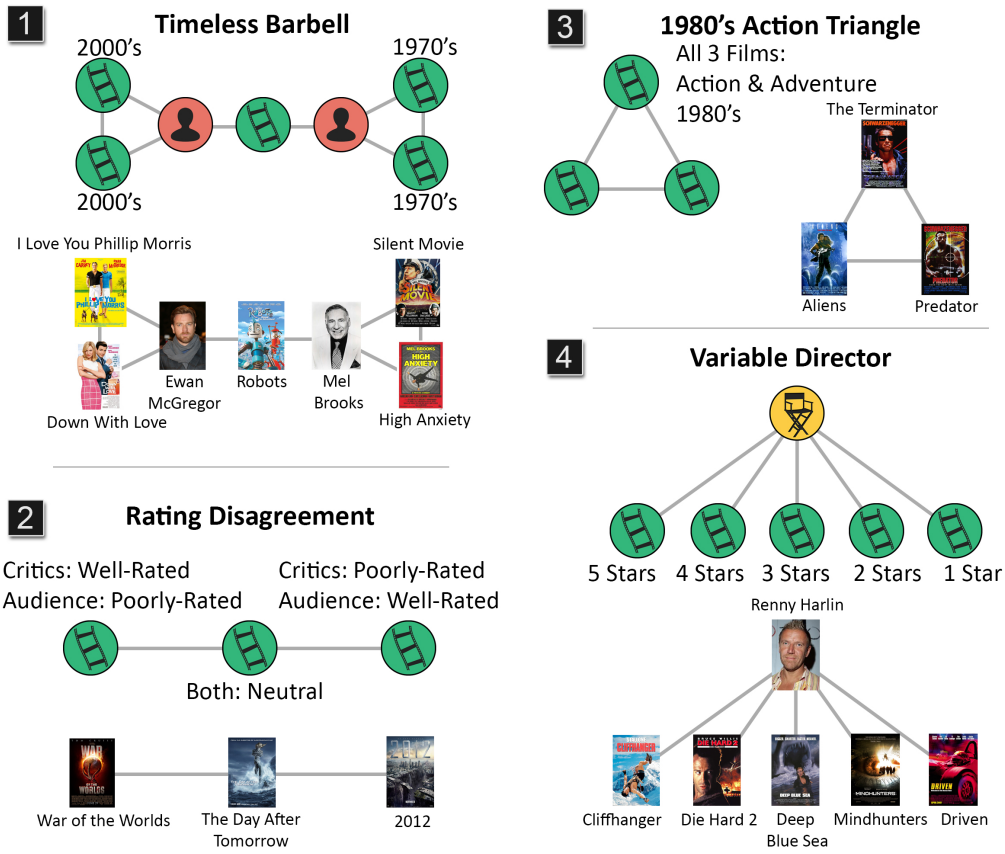


Figure 5.11: Four queries used in scalability experiments, which blend both structure and feature constraints. **The Timeless Barbell** (1) , shows a query for an actor in two similar movies from the 2000's, who co-starred with an actor in two similar films in the 1970's. **Rating Disagreement** (2) shows a chain of three similar films where the critics disagree with the audience on what good movies are. **1980's Action-Triangle** (3) shows three similar action films from the 1980's. **Variable Director** (4) investigates directors whose filmography contains the whole spectrum of critics' ratings.

Rating Disagreement, in Figure 5.11.2, demonstrates a chain of three similar films. The first film had better critics' scores than audiences, in the second film the critics and audience agreed on a neutral score, and in the third the audience thought the film was better than the critics. A few more examples are; Space Cowboys, Lifeforce, and Blade Trinity; The Bunker, Devil, Saw; and Antz, Bee Movie, and Little Chicken.

We wanted an action-packed trio of films so we created the *1980's Action Triangle* query. This three-clique, shown in Figure 5.11.3, has three similar action movies all from the 1980's. A few other matches we found are: Back to The Future, The Goonies, and

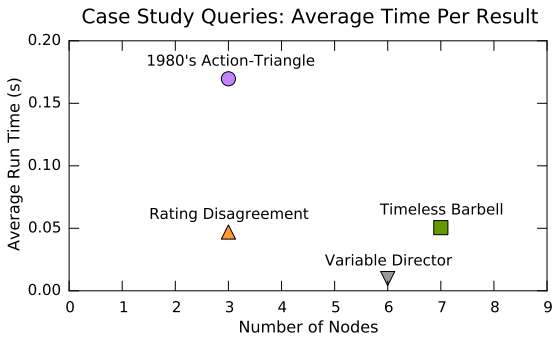


Figure 5.12: Run times for the queries from VISAGE's case study. These queries represent examples with both structure and constraints, ranging from 3 nodes up to 7 nodes. The slowest query is the **1980's Action-Triangle**, because it's a constrained 3-clique which is much harder to locate than the other examples. All queries finish in under a third of a second.

Time Bandits; The Beastmaster, Dragonslayer, and Krull; and Indiana Jones and the Last Crusade, Raiders of the Lost Ark, and The Jewel of the Nile.

Ever been curious about directors whose work spans the entire spectrum of film ratings? We created the *Variable Director* example query (see Figure 5.11.4) to find whose filmography spanned the ratings spectrum. A few other variable directors are: Robert Rodriguez, Wes Craven, Francis Ford Coppola, and unsurprisingly M. Night Shyamalan.

The difference likely arises from each of the triangle's nodes having two constraints and all being interconnected, which is much harder to fulfill than the Timeless Barbell's constraints. All results were processed fast enough to be used interactively.

5.8 Conclusion & Future Work

In this chapter we presented VISAGE, a system built using recent innovations in graph-databases to support the visual construction of queries, from abstract structures to highly conditioned queries. VISAGE relies on an interaction technique for graphs called *graph-autocomplete* that guides users to construct and refine queries, preventing null-results. We hypothesized that visual graph querying with VISAGE would be faster for generating queries than the Cypher querying language. We demonstrated this with a twelve-

participant, within-subject user study. The study showed that VISAGE is significantly faster than conventional querying for participants with or without familiarity to Cypher.

VISAGE offers users a visually supported, code-free solution to graph querying that helps guide the user towards queries with results. Currently we do not support graph sub-queries, unions and intersections (of graph results), aggregations, shortest-paths, and edge attributes. This chapter has revealed additional challenges and potential new questions for the community. How can inexact or approximate querying be used to aid query construction and refinement; and how best to visualize the uncertainty inherent in approximate results [10, 12]? We hope VISAGE will spur continued interest in visual graph querying.

Thrust III

Visualizing and Exploring Subgraph Matches

OVERVIEW

After an analyst has navigated and explored a graph with Thrust I, formed visual graph queries and refined them in Thrust II, they will be faced with numerous subgraph matches. 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. Currently, few graph querying systems offer more than tables, lists, or basic graph layouts for exploring results. A set of subgraphs from a graph query will likely be rich with additional information, either from the nodes themselves or from the underlying topology in which they were found. We propose several novel visualization techniques, designed to overcome the challenges of summarization and comparison across a set of results.

In this Thrust, we introduce VIGOR, a system which utilizes algorithms, interaction design, and visualization techniques to help summarize and compare large numbers of subgraph results (Chapter 6). We introduce a new method to produce high-level summaries of results through feature-aware result-subgraph embedding.

CHAPTER 6

VIGOR: INTERACTIVE VISUAL EXPLORATION OF GRAPH QUERY RESULTS

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 [178]. Or in online auctions, analysts may want to uncover "near-bipartite cores" formed among fraudsters and their accomplices [84]. While there is significant research interest and development in graph algorithms, database management systems and even visual graph query construction techniques [188, 83, 179], 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 chapter, 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 [86], cybersecurity [84], social network analysis [92], to finance [178].

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

Chapter adapted from work under review at VIS'17 [Paper Link]
Poster Version of the Work at IEEE VIS'16 [Paper Link] [189]

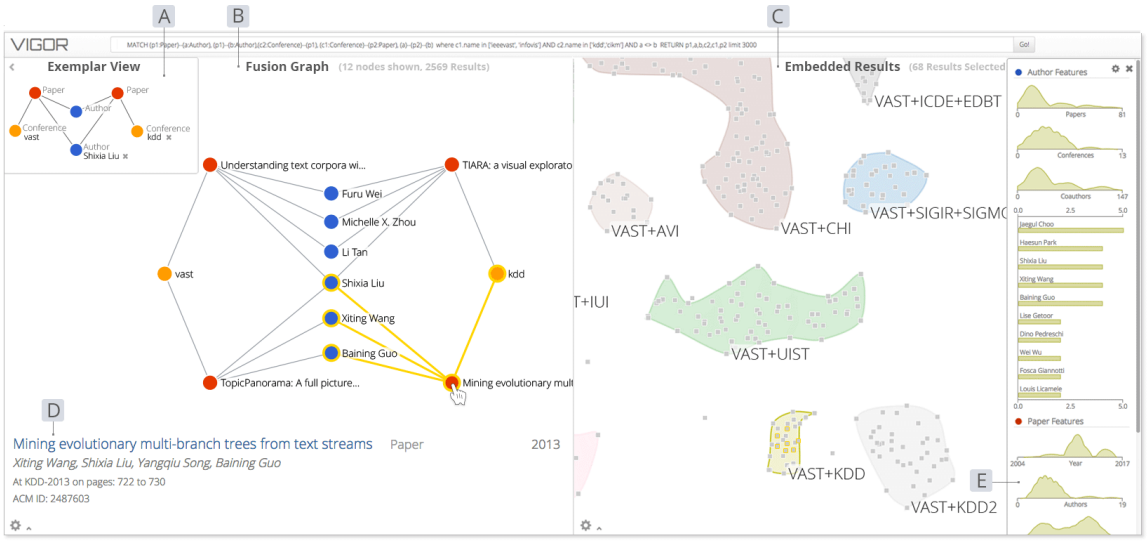


Figure 6.1: A screenshot of VIGOR showing an analyst exploring a DBLP co-authorship network, looking for researchers who have co-authored papers at the VAST and KDD conferences. (A) The Exemplar View visualizes the query, and (B) the Fusion Graph shows the induced graph formed by joining all query matches. Picking constant node values (e.g., Shixia) in the Exemplar View filters the Fusion Graph. (C) Hovering over a node shows its details. (D) The Subgraph Embedding embeds each match as a point in lower-dimensional space and clusters them to allow analysts to see patterns and outliers. (E) The Feature Explorer summarizes each cluster's feature distributions.

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 6.2). 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 [190].

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 [191, 192]. The important contributions of VIGOR include:

- **Exemplar-based interactive exploration.** VIGOR simultaneously supports *bottom-up* sensemaking [190], where an analyst starts with a specific result and relaxes con-

straints 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 6.1A). 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 [193, 190]. We introduce an algorithm to group results by node-feature and structural result similarity (Figure 6.1C) 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 6.1B) 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

[1]DBLP Website: `http://dblp.uni-trier.de/`

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.

Figure 6.2: (A) Neo4j, a commercial system, displays subgraph matches in a long table. One match with three nodes is shown here, each gray box describes one nodes' features. (B) VISAGE [179] displays subgraph matches in a list, without revealing connections among results. Even for modest sized queries, these conventional approaches require significant scrolling, and cannot easily reveal broader patterns and relationships among matches.

6.1 Introducing VIGOR

To illustrate how VIGOR works in practice, we will briefly cover an overview of the system's components (in Section 6.1.1) and an illustrative scenario where we explore co-authorship in a DBLP network.

DBLP Dataset. In this chapter we utilize a real co-authorship network drawn from a subset of DBLP's computer science bibliography data. The undirected, unweighted network

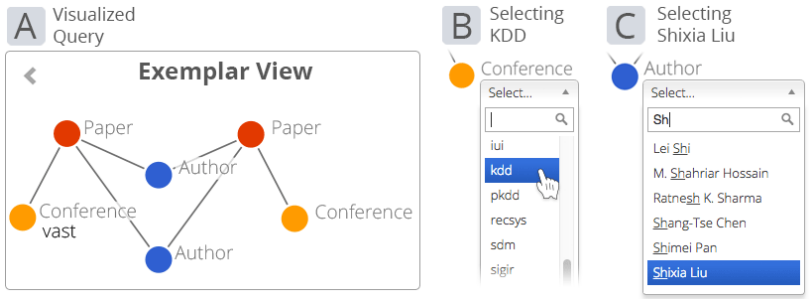


Figure 6.3: Exemplar View displaying a query seeking researchers who have coauthored papers at two different conferences. (A) The analyst starts with only the *structure* of the graph query, then incrementally adds node value constraints to narrow in on specific results, (B) first by choosing KDD, which (C) narrows down the remaining choices for authors.

contains 59,655 authors, 48,677 papers, 7,236 sessions, 417 proceedings, 21 conferences and 1,634,742 relations from the data mining and information visualization communities. We will use this network in both the illustrative scenario (Section 6.1.2) and in our user study (Section 6.4.1).

6.1.1 VIGOR Interface Overview

The VIGOR user interface is composed of four main areas (Figure 6.1). The *Exemplar View* at the top (Figure 6.1A) visualizes the user's textual graph query (entered into the text form at the top of Figure 6.1) and supports quick filtering by value. The *Fusion Graph* (Figure 6.1B) displays an induced graph of all the result subgraphs from the query, quickly demonstrating which nodes appear often and with which other nodes. The *Subgraph Embedding* view (Figure 6.1C) summarizes all the results by reducing each result into a square, gray glyph and clustering them (colored, concave clusters) based on feature similarity. Analysts are free to create, name, and compare their own clusters. Clusters are compared in the *Feature Explorer* view (Figure 6.1E), which provides summary distributions of each node type included in the results. The goal of the VIGOR interface is to enable analysts to detect underlying patterns in their result set as well as explore individual values with as little tedium as possible. The synergy of these techniques across our three views enables analysts to explore their query results with ease.

6.1.2 Illustrative Usage Scenario

We demonstrate how VIGOR works with an illustrative example exploring a cross-conference co-authorship query. Imagine an analyst, Alexis, is interested in finding authors and papers that bridge the information visualization and data mining communities. This scenario demonstrates some of the interactions and major features in VIGOR.

Because Alexis wants to learn about papers, conferences, and authors, she begins with a query looking for an author who has published two papers with a co-author, where the pa-

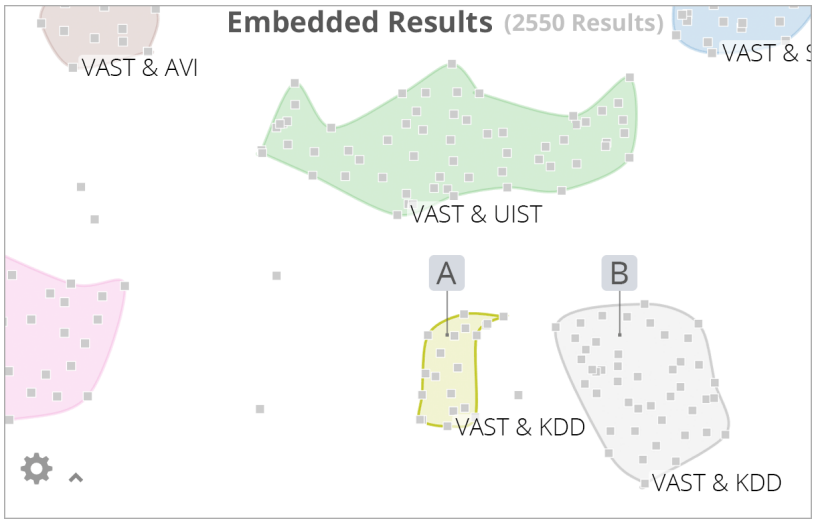


Figure 6.4: The Subgraph Embedding provides an overview of the results through the feature-aware subgraph embedding, where results are displayed as points in two dimensions based on node feature similarity. We see the clustered results of a query seeking two co-authors of two papers at VAST and another conference (shown in Figure 6.3). Nearby clusters (A) and (B) both contain VAST and KDD papers, the features of which are compared in Figure 6.6. Cluster labels are customized by the analyst during exploration.

pers were published 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 6.3A).

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 6.4) are placed spatially close together by the feature-aware subgraph result embedding and clustering (Section 6.3). To help her differentiate among groups, VIGOR uses a density-based clustering technique [194] 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

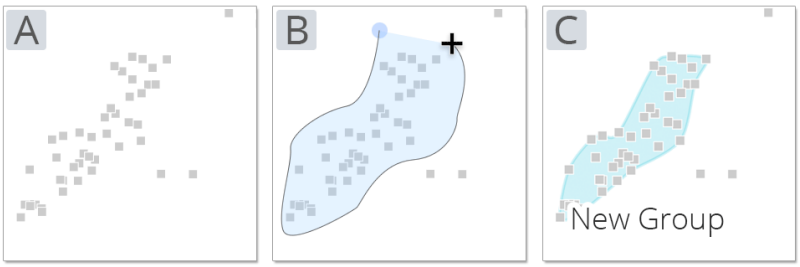


Figure 6.5: (A) Starting from a group of results, (B) an analyst lassos the desired results. (C) A concave hull is established forming a cluster with the points. Cluster can be used to: filter the Fusion Graph and compare features and node values in the Feature Explorer.

create and name her own clusters by lassoing groups of points (Figure 6.5A-C).

She shift-right-clicks two neighboring clusters (Figure 6.4A and 6.4B) to compare them in the Feature Explorer (Figure 6.6). The Feature Explorer shows common node values and feature distributions for each node type included in the clusters, similar to [149, 195]. 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 6.6) 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 6.4). 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 6.6-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 searchable dropdown menu with the matching conferences (Figure 6.3B). She selects KDD. When she clicks on an author node in Figure 6.3A, 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

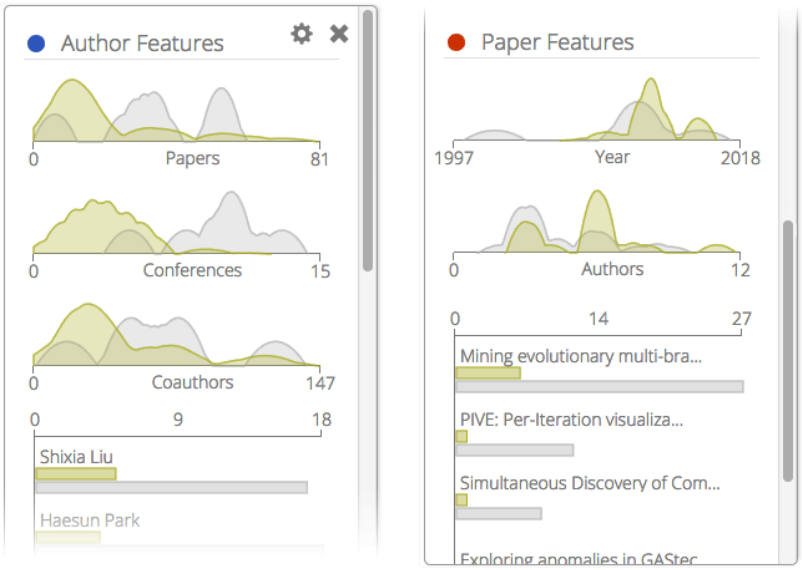


Figure 6.6: The Feature Explorer shows common node values and feature distributions for each node type included in two clusters (A and B in Figure 6.4). The features for each node type in the Fusion Graph view are summarized as distribution charts. The bar chats show the top-k most common values, including those shared between the selected clusters.

(Figure 6.3C).

Alexis' selection in the Exemplar View filters the Fusion Graph (Figure 6.7), 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.,

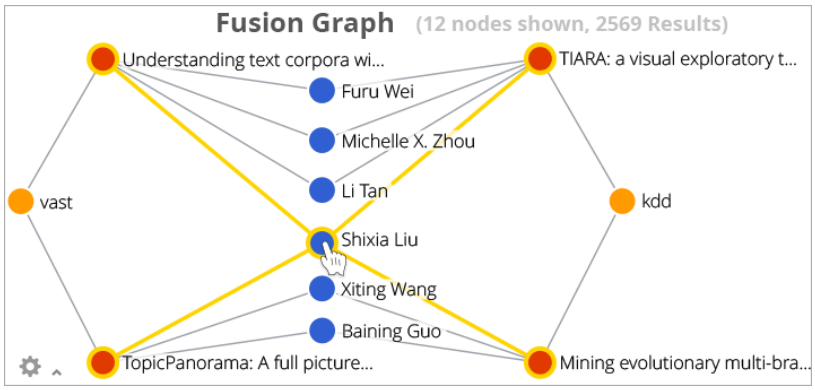


Figure 6.7: Shixia Liu's papers and co-authors who have published papers together at VAST and KDD. The Fusion Graph view shows an induced subgraph of all the combined results from the original query, which can be filtered from either the Subgraph Embedding or the Exemplar View.

Michelle, Furu, Li, Xiting, and Baining in Figure 6.7). 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.

6.2 Core Design Rationale

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

6.2.1 Leveraging Examples: Bottom-Up Exploration

Starting with low level details is often referred to as a bottom-up sensemaking [190, 10]. Starting from a known example can greatly improve the development and understanding of a query [58]. We designed the Exemplar View (Figure 6.3) 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

6.2.2 A View From Above: Top-Down Exploration

High level overviews, like the Subgraph Embedding (see Figure 6.4), have proven useful in visualization models for sensemaking in other datasets [193, 10]. 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 [109]. The details of our graph embedding algorithm are discussed further in Section 6.3.

6.2.3 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 [195] and node-feature distributions [149] have proven a valuable way to compare results. To compare these features, we created the Feature Explorer (Figure 6.6), 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.

6.2.4 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.

6.3 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

6.3.1 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 6.8):

- *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 [109]; however, our approach differs in

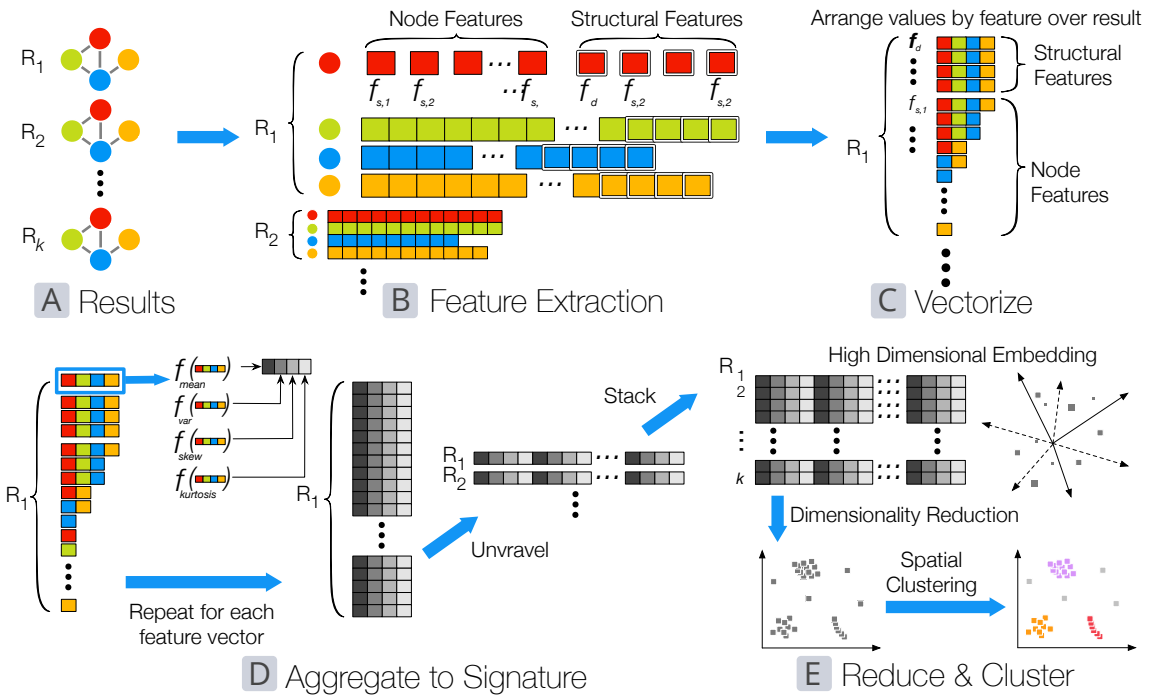


Figure 6.8: Given a set of k results (A), we first extract topological features (B) from the neighborhood around each result in the underlying graph, which are combined with features from each node in the result. Next the values are rearranged by feature (C). These feature sub-vectors are run through the moment 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).

several key areas outlined below.

Extract Features We use both the node-features f_s and a small set of topologically extracted features f_t as inputs to our embedding (Figure 6.8A and 6.8B). There are many different ways to extract features from a graph. We started with the structural features from [109] and NetSimile, [106], 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 [106] 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

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

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

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

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

$$|N(ego(i))| = \left| \bigcup_{j \in N(i)} N(j) \right|,$$

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

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

Vectorize Each node of a result now has the four structural features from 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 6.8C). 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 6.8B).

Aggregate & Normalize To solve the issue of uneven lengths of the per-result feature vectors we convert them into a signature (see Figure 6.8D). 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 distributions to reduce the feature vectors into a fixed length signature. We use the first 4 moments: mean, variance, skewness, and kurtosis. For robustness we cannot use the mean and variance alone, because both structural features and node-features may not be normally distributed. The skew moment measures the lopsidedness of a distribution while the kurtosis gives a measure of how heavy the tail of the distribution is. We perform these for each feature vector per result and wrap them into a single array, yielding a new signature of length $4 \cdot (|f_s| + |f_t|)$, where f_s and f_t are the sets of features from the nodes and the structure respectively.

Reduce & Cluster We then perform dimensionality reduction to reduce the dimensions to two (see Figure 6.8E). There are many dimensionality reduction techniques both linear and nonlinear. We default to Principle Component Analysis (PCA) [110], but allow the analyst to choose among kernel-PCA [196], multidimensional scaling (MDS) [114], and t-Distributed Stochastic Neighbor Embedding (t-SNE) [116]. We chose to offer PCA first due to its fast performance and simple linear nature.

Both MDS and t-SNE allow arbitrary distance functions rather than the Euclidean distance. For both MDS and t-SNE we compute the Canberra distance (or weighted L_1 Manhattan distance) [197] rather than the Euclidean distance. We chose Canberra because it is sensitive to small changes near zero, which helps preserves small distances in the final reduction. It has also performed well on real datasets [123].

We perform clustering on the dimensionally reduced points (see Figure 6.8E). There are many density-based clustering algorithms like DBSCAN [198] or OPTICS [194]. We use OPTICS to perform our density-based clustering, because it performs better on clusters with different densities [194]. Because the choice of ϵ greatly affects the resulting clusters, we allow the user to adjust the value via a slider. The cluster information in encoded as colors in the Subgraph Embedding.

6.3.2 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 by 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 reductions 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.

6.4 Evaluation

We performed a two-part user evaluation of VIGOR (Section 6.4.1). In the first part, we compare VIGOR against Neo4j, a leading graph DMBS. 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[2], the leading security company, to identify blindspots in the understanding of critical security incidents. In Section 6.4.2, 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 6.1.

Table 6.1: Graph datasets used for evaluation: DBLP dataset for user study; cybersecurity dataset for real-world application to discover security blindspots.

Network Type	Node	Edges	Node Types
DBLP	$115,989$	$1,543,792$	5
Cybersecurity	$17,651$	$384,172$	3

[2]We invited our Symantec collaborators to join as coauthors of this chapter.

6.4.1 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 [83, 179, 58]; 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.

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.

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 one, or if time ran out. Incorrect answers were recorded for each task, including if they ran out of time before answering.

Once a participant had completed all the tasks with a system, they would repeat the same process with the next system (including the system demonstration). Participants were not informed which system, if either, was developed by the examiner. After a participant had completed both comparative tasks, we asked them to complete Part II, the think-aloud exploration study. At the end of the study, participants completed a questionnaire that asked for subjective impressions about each software system.

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 [92, 76, 93]. The tasks from Task Set A (shown in Figure 6.9) 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.

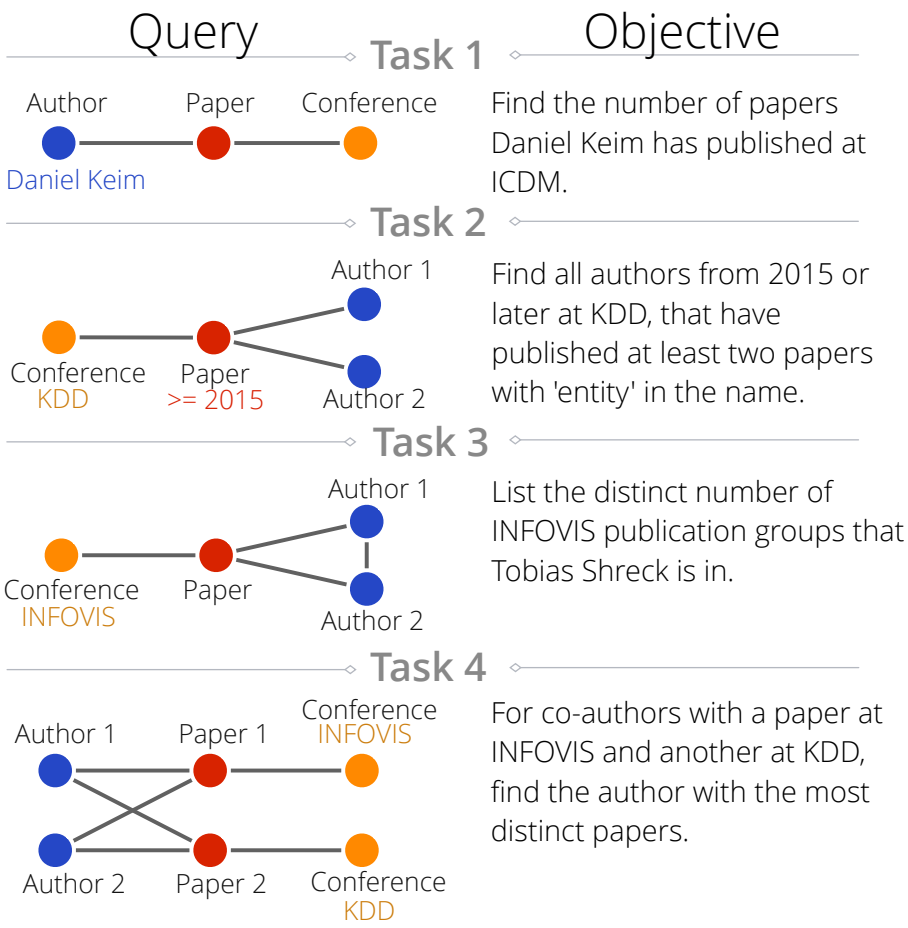


Figure 6.9: VIGOR user study comparative tasks. These tasks were provided to create the result sets used in Part I of our user study. Both task sets utilized the same query topologies, but different values, carefully selected to have the same number of results.

4. 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

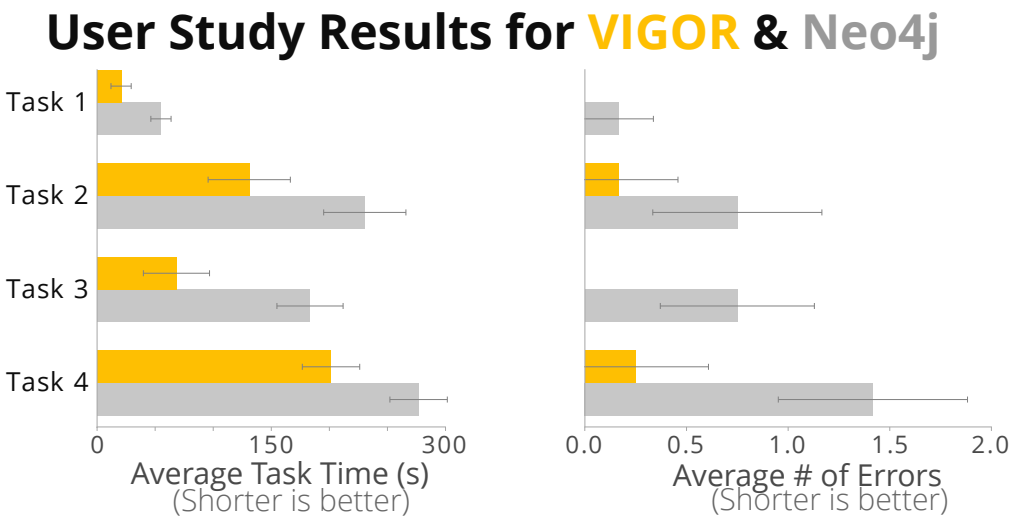


Figure 6.10: Average task completion times and error rates for VIGOR (yellow) and Neo4j (gray). VIGOR is statistically significantly faster across all tasks. Error bars represent 95% confidence interval.

counterbalanced the possible confounding factors (e.g., one group is *(*VIGOR + *Task Set A) then (Neo4j + Task Set B)*).

Quantitative Results We analyzed task completion times usinng mixed-model analysis of variance (ANOVA) with fixed effects for *software, software order, task set*, and a random effect across *participants.* Mixed-model ANOVA improves over conventional ANOVA as errors are calculated per-subject.

Our task completion times were measured over all combinations of software order and task set. The experiment was successful as the only statistically significant effect was from software system. Figure 6.10-left demonstrates the average time per task in our study. The software effect was significant for each task: task 1 ($F_{1,11} = 29.79,\ p < 0.0003$), task 2 ($F_{1,11} = 41.02,\ p < 0.0001$), task 3 ($F_{1,11} = 33.68,\ p < 0.0002$), task 4 ($F_{1,11} = 23.89,\ p < 0.0006$). Only task 3 ($F_{1,11} = 12.27,\ p < 0.0057$), and task 4 ($F_{1,11} = 19.6,\ p < 0.0013$), had statistically significant error rates. This is expected as the error rates for the first tasks were very low. The second task in Task Set A came close to significance with ($p < .048$), likely arising from slightly higher number of edges in the induced subgraph than in Task

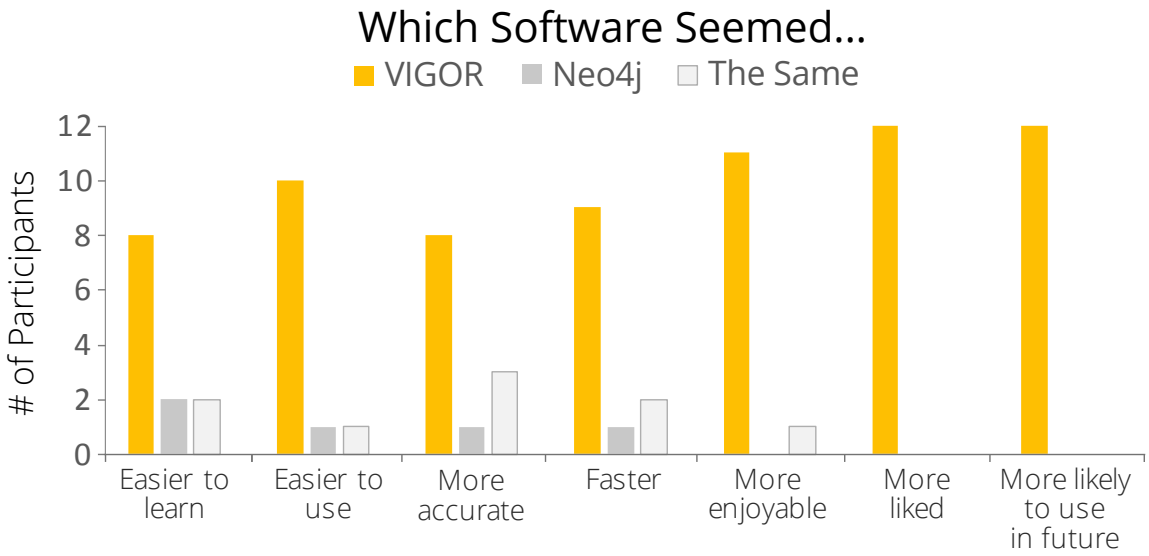


Figure 6.11: Participants were asked to qualitatively compare each system at the end of each trial. Overall, they felt that VIGOR was better than Neo4j in all of the 7 aspects asked.

Set B. Participants were both significantly faster and less prone to error with VIGOR versus Neo4j.

Subjective Results At the end of the study we asked participants to rate various aspects comparing both systems using Likert scales. Participants felt that VIGOR was better than Neo4j for all 7 aspects asked (Figure 6.11). One participants stated, "I enjoyed the clustering features of VIGOR, allowing the user to quickly compare variables (Year, etc.) about any possible combinations of groups.". The participants enjoyed using VIGOR more than a Neo4j and reported that our system was: easier to learn, easier to use, and more likeable overall; although this is a common experimental effect, we find the results encouraging.

Part II: Think-aloud Exploration Study

After the comparative tasks were completed, all participants were asked to perform a think-aloud exploration study. We chose to separate this part of the study from Neo4j as it tests new features that are not present in Neo4j's interface. This part of the study was not timed.

Our goals for the think-aloud study were:

- **Feature interactions:** were our features were working well together, and whether

VIGOR met their basic exploration needs.

- **Identify usability issues:** were features usable and if they coordinated in beneficial ways during their exploration.
- **Feature application:** what techniques participants would use with VIGOR and whether its functionality would help streamline their analytics workflow.

High-level Objectives We provided participants with a pair of scenarios and high-level objectives to complete. We asked participants to imagine themselves as researchers interested in:

1. the features from all papers by Jiawei Han or Christos Faloutsos at PKDD and SIGMOD; and
2. understanding the outlier results (results distant from a cluster) for co-authors of papers at VAST and KDD or INFOVIS and KDD.

We provided the queries for both tasks. Participants were free to use any features of VIGOR and ask questions during the objectives. We chose the above objectives, because they are common in graph analysis [199, 200].

Key Observations During the first objective, 6 participants began their exploration by searching for PKDD and SIGMOD using the Exemplar View to find the conferences. Another 4 of the participants went directly to using the Fusion Graph to highlight results in the Subgraph Embedding by hovering over specific conferences. The remaining 2 participants used the Feature Explorer's conference type to investigate which clusters contained PKDD and SIGMOD conferences. For 4 of the 12 subjects they had considerable difficulty with their first few lassoing attempts, often completely missing the desired nodes. Only 2 participants failed to adequately complete the objective.

In the second objective, 10 participants started by creating new clusters by lassoing groups of outliers to compare them against the existing clusters. The remaining 2 used

the Fusion Graph to highlight results in the Subgraph Embedding for particular nodes. Of 12 participants 3 reported that they had not found any satisfactory explanations for outliers, while the remaining 9 either found specific papers or features not present in the cluster. One participant correctly commented that several of the outliers arise from single-author papers, because multi-author papers have a higher chance of being repeated across the results (and therefore have a higher chance of being similar to other results). Overall participants performed very well using the coordinated views in VIGOR.

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 [179]. Even the queries provided were

challenging to many participants, as demonstrated by the high error rates in Task 4.

We were pleasantly surprised to see that participants were able to use and compare features using the cluster-based distributions in the Feature Explorer (Figure 6.6) and that they could use when comparing more than two clusters.

While our evaluation was very positive, the real-world scenarios and initial queries of analysts would be ad hoc. We plan to study this case and better understand how VIGOR can handle tasks in less planned situations. For example, how would analysts utilize the Subgraph Embedding for significantly different domains, transportation networks, intelligence, bioinformatics?

6.4.2 Real World Application: Discovering Cybersecurity Blindspots

We collaborated with three security researchers from Symantec to identify blindspots in their company 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 6.12A), 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 6.12B).

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

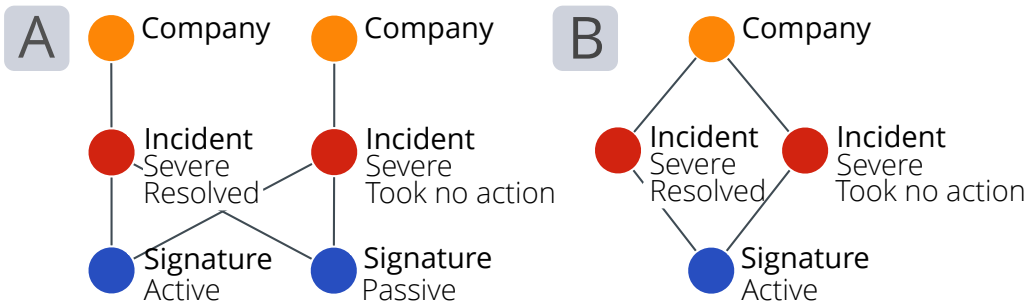


Figure 6.12: Example queries from our exploration of cybersecurity blindspots. Query A reveals companies ignoring critical security incidents that their peers resolve, whereas Query B extracts companies responding inconsistently to critical security incidents.

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 immediate investigation and resolution, but frequently are not.

Query 1 - Comparing Company Incident Behavior Our first query (see Figure 6.12A) 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 6.13), 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.

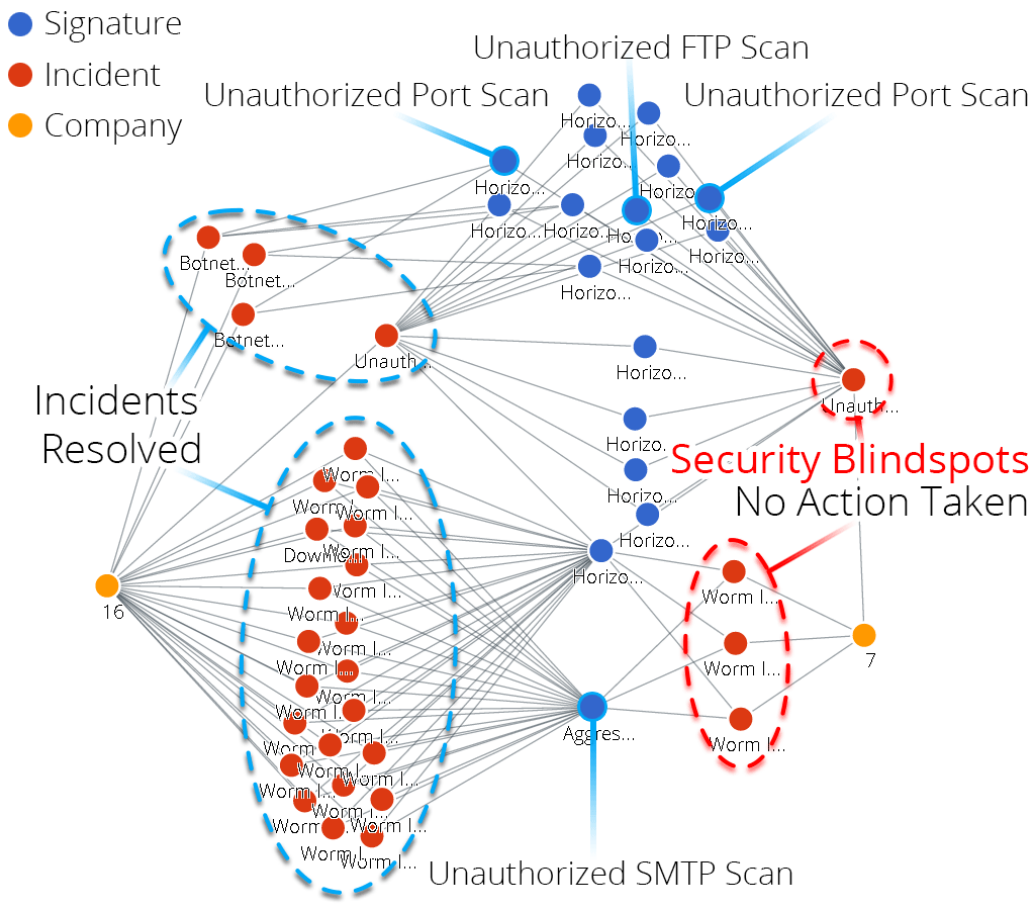


Figure 6.13: Results of our first blindspot detection query (see Figure 6.12A). VIGOR identifies Company 7's blindspots with evidence that Company 16 is faced with similar security incidents and takes them seriously.

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 6.12B, 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 6.14A and 6.14B. 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 6.14A could be spreading by means of the unmitigated unauthorized internal vulnerability scans that happened in a similar timeframe. Similarly, internal machines still

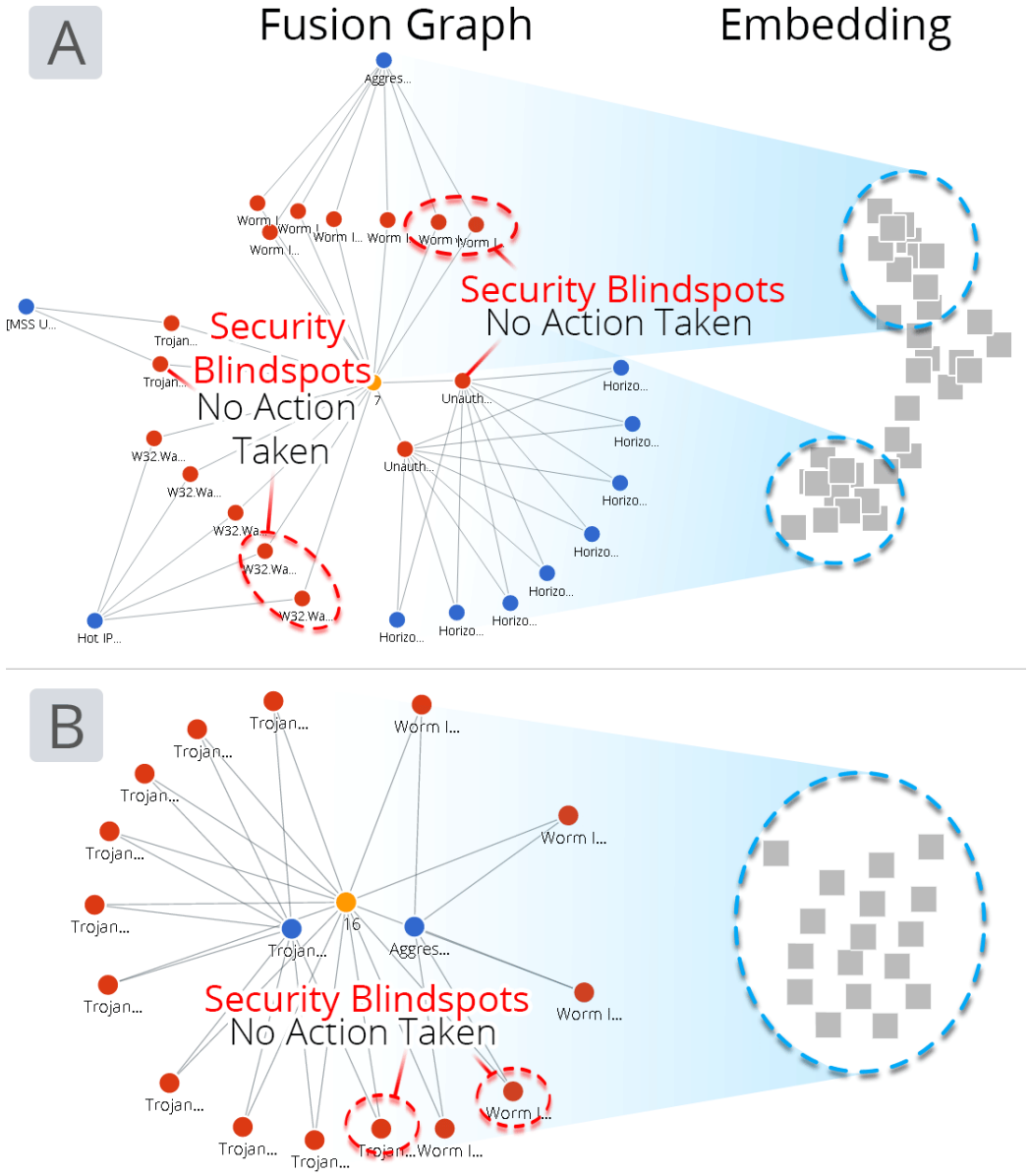


Figure 6.14: Results of our second blindspot detection query (see Figure 6.12B). VIGOR identifies companies that respond inconsistently to critical malware-related security incidents, while majority of the incidents are resolved (circled in green), some received no action (circled in red).

infected by the trojan malware of Figure 6.14B 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 6.14A and 6.14B both highlight the way in which the Feature Embedding is able to cluster related security blindspots in two dimensions for efficient perusal.

6.5 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_{neigh}) 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.

6.6 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.

Conclusions

CHAPTER 7
CONCLUSIONS

The era of big data is upon us. Network datasets are growing incredibly fast; yet, making sense of these networks remains a fundamental challenge. This thesis advocates *combining* data mining and visual analytics research to guide researchers and practitioners when making sense of network data. The ideas, techniques and systems previously discussed promote exploration of large network data without the need of custom querying languages.

7.1 Contribution Summary

We make three important contributions to network analytics research:

- **Thrust I: Adaptive Local Exploration** Large graphs pose a significant challenge to visual analytics approaches, because the large number of nodes and edges in a graph often visually occlude each other. Even modest sized graphs are hard to utilize as a starting point for exploration. We propose to use adaptive local exploration to overcome the challenges of visual scale when facing massive graphs. In **Thrust I**, we guide analysts towards nodes and node-neighborhoods with the FACETS system (covered in Chapter 3); enabling analysts to adaptively explore large million-node graphs from a local perspective.

- **Thrust II: Interactive, Visual Graph Query Construction and Refinement** The foraging from Thrust I: Adaptive Local Graph Exploration, may yield interesting subgraph patterns. Other similar subgraphs may exist, but may be topologically far from the currently explored region. Graph querying extends the reach of local exploration once a pattern is known (or even partially known). Constructing and refining graph queries requires complex querying languages [4, 5, 6, 7]; however, we show

that queries can instead be formed via a visual, interactive system. Querying is often an iterative process that can benefit greatly from visual aids [8, 9]. Making such a system requires both algorithms and visualization working in tandem, as few algorithmic approaches have been designed for both interaction and visualization. This thrust focuses on algorithms to locate matches and techniques that help analysts construct and refine visual graph queries.

- **Thrust III: Visualizing and Exploring Subgraph Matches** After constructing a query an analyst will receive numerous subgraph matches (which we will call results). 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, various node features, and flexible structure across queries. Our system, VIGOR, combines scalable algorithms and interactive visualization techniques to help summarize and compare large numbers of subgraph results (Chapter 6).

7.2 Contributions

Our contributions to *data mining, visual analytics,* and importantly their *intersection* follow:

- **Algorithms.** We introduce FACETS (originally called ADAPTIVENAV) (Chapter 3) which utilizes fast, dynamic rankings of node neighborhoods to lead analysts towards interesting content. We design and develop a highly scalable algorithm in MAGE (Chapter 4) for approximate subgraph matching on large networks with node and edge attributes, wildcards, and multi-attributes. MAGE uses a novel approach based on the linegraph transformation to embed edge attributes. In VISAGE (Chapter

5), we create algorithms to support dynamic interactive graph querying in real time by transforming visual queries into written queries in two querying languages. In our VIGOR system (Chapter 6), we introduce a novel, feature-aware subgraph result embedding. This embedding uses semantic node-feature data as well as topological data when creating a high-dimensional result embedding.

- **Systems.** We contribute our algorithms to the research community as planned open-sources projects for approximate subgraph matching (MAGE), adaptive local exploration (FACETS), interactive visual graph query construction and refinement (VISAGE), and exploratory visualization of subgraph results (VIGOR).

- **New Metrics to Determine User Interest.** In FACETS, Chapter 3, we contribute two new metrics based off of the Jensen-Shannon divergence of feature-neighborhoods: (1) for subjective user interest modeling and (2) for measuring a nodes rarity and surprisingness against a network.

- **New Adaptive Graph Exploration Paradigm.** We contributed a survey that summarizes and discuss many challenges and opportunities for new network analytical systems (Chapter 2). FACETS is a new step towards an adaptive-exploration paradigm in visual analytics, exploiting the data generated during interactions between a software system and a user to aid the exploration of data.

- **New Interactive Graph Query Construction and Refinement via Graph Autocomplete.** VISAGE (Chapter 5) represents a major step towards interactive, visual graph querying. It provides a new, highly-usable querying paradigm, that uses simple drag-and-drop interaction to construct complex queries, rather than having to learn a querying language VISAGE introduces a novel interaction technique called *graph-autocomplete*, which guides analysts away from over-specifying their queries. VISAGE outperforms conventional querying writing and refinement in a laboratory study for both expert and non-expert subjects.

- **Visual Result Exploration** VIGOR introduces a new overview mechanism to visualize large numbers of subgraph results (Chapter 6). It also employs an exemplar filter to explore the results by filter down to only results with familiar nodes.

- **Scalable Interactive Tools.** Our interactive systems advance the state of the art, by facilitating real-time discovery and exploration in graphs with millions of edges (e.g., VISAGE and FACETS). Empirical runtime analyses demonstrated FACETS s practical scalability on large real-world graphs with up to 5 million edges, returning results in fewer than 1.5 seconds. We demonstrated realtime querying with VISAGE, with sub-second querying times on a real Rotten Tomatoes movie graph with over 170 thousand relationships.

7.3 Future Research Directions & Transition to Practice

This thesis has taken major steps in improving the analytical capabilities of analysts exploring networks. Through developing our systems and interacting with practitioners, study participants, and querying experts we have identified three interesting avenues of future research.

Faceted condition generation for iterative query construction Faceted search (sometimes called faceted navigation or faceted browsing) is a technique for accessing information via a faceted classification system in which users explore information by applying filters (e.g., the filter-bank provided by Amazon or almost any popular online shopping site). The facets in a faceted classification system are typically features or properties that enable information to accessed an ordered in a variety of ways rather than a taxonomic order [201]. The information retrieval and human-computer interaction fields have performed significant research into faceted search systems [201, 202, 203, 204].

These techniques could enable multi-dimensional exploration of both query results as well as the generation of new query conditions. The interface techniques used in prior

research could be the foundation for new visual techniques when exploring large numbers of subgraph matches. Each chosen facet would narrow the result-set and could be simultaneously generated into a query condition programmatically. This exploration technique would likely be very successful from a practical standpoint, as faceted search interfaces in internet shopping are ubiquitous.

Exploration and comparison of novel subgraph embeddings Recent advances have introduced a large number of new subgraph embeddings. Grover et al. released a technique called *node2vec*, a graph embedding that smoothly blends local and global characteristics when creating a continuous, high dimensional representation [107]. Recently *subgraph2vec* was also suggested, which learns latent representations of subgraphs in a continuous, high dimensional space for use in deep-learning applications [108].

In many applications the location of a subgraph in the network plays an important role. Our feature-aware subgraph embedding only uses 1-hop neighborhood features from within the underlying network, node2vec and subgraph2vec could be used to provide additional context from the network when embedding results. This could be used to provide summaries that take into account aspects of results' positions in the broader network.

These methods also promote and enable the use of other data mining techniques to subgraphs. This vastly increases the possibilities for prediction, estimation, recommendation, and even simulation. Additional research is needed to compare and contrast how well these embeddings work for different tasks (e.g., prediction).

Visual methods for providing complex node and edge conditions A common aspect of querying is providing conditions on nodes and edges (e.g., a film node with condition: $year = 1988$). These conditions allow analysts to quickly filter down results. However, the types of data associated with nodes and edges can vary greatly (simple integers, text, code, images etc.), making condition generation a broad and challenging task.

Many querying languages allow complex expressions including conventional mathematical operators, set operations, aggregations, regular expressions and more. These condition-expressions make querying languages incredibly flexible and expressive. Our VISAGE work provided a prototype method for visually constructing node conditions over categorical features, but does not cover the full breadth of data types and expressions.

Previous research has investigated visual approaches to construct and represent only a few types of condition-expressions. Vi-xfst, a development tool for Xerox finite-state tool from computational linguistics provided a visual mechanism to create regular expressions [205]. SWYN, See What You Need, is a system for string-matching regular expressions that utilizes and suggests visual widgets to construct the expressions [206]. Lixto, visual system for information extraction (specfically frrom html data to structured xml), has module for visual construction of basic expressions [207]. These provide the starting point for making a deeply expressive visual querying language.

CHAPTER 8

APPENDIX A

8.1 Additional Observations

8.1.1 On Providing Distributions to Novice Participants

Our pilot study of FACETS demonstrated that participants typically did not understand the feature histogram comparisons in the hidden neighbors menu. In the laboratory study for VIGOR, participants were more capable of using the distributions to compare groups. We provide an example of the FACETS layout in Figure 8.1 on the left and an example of VIGOR's on the right. This is likely driven by the design choices we followed in each:

Figure 8.1: The distributions used by (left) FACETS and (right) VIGOR.

- **Distribution Appearance:** FACETS utilized MDL-binning from [162] to bin the distribution, while VIGOR used kernel density estimation with a Gaussian kernel. While MDL-binning chooses information-theoretically optimal bin sizes, they often left participants confused about meaning. Surprisingly little research has investigated how well participants make sense of different distribution visualizations. In FACETS case the variable-width bins increased visual complexity.

- **Comparison Mechanism:** FACETS compared the distributions by showing stacked MDL-bins, where the number of values in the bin was encoded with its saturation.

Stacked barcharts are notoriously hard to read [208] and often are used to represent parts-of-a-whole. When combined with the variation in colors likely further confused participants.

8.1.2 Expressive Querying and Non-experts Constructing Visual Queries

The breadth of possible queries is immense; however, many of the ideas needed to make complex queries require expertise in querying. Because expert-only studies are notoriously challenging to scale, we chose to test general query construction and refinement with non-experts. We designed VISAGE with a subset of the expression-rich Cypher querying language; we only developed visual techniques for expressions from which the returned results were subgraphs. This design specifically excludes: subgraph set operations, aggregations, custom paths, and potentially others language features. To test these features and more challenging queries (e.g., computing the transitive closure over a graph) we would need experts.

When we performed our user study of VISAGE, we picked from a pool of Georgia Tech students. Before participating in the study they were asked to complete an entrance questionnaire. We asked about: *their familiarity with SQL and DBMSs*, *their knowledge of graphs and networks*, and *their experience with the Cypher querying language.* The distributions of participants self-reported answers are shown in Figure 8.2.

Our goal with this study was to assess the query construction and refinement techniques

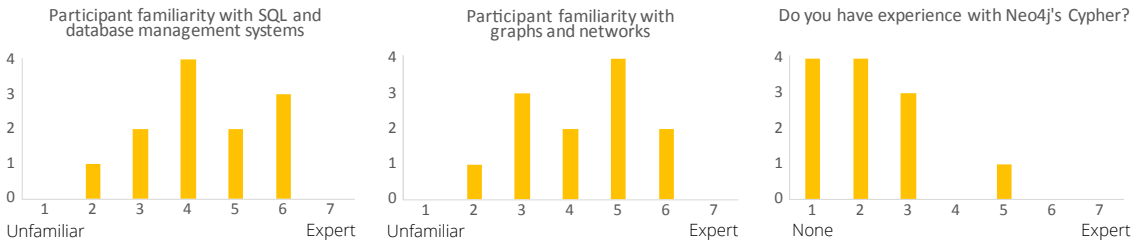


Figure 8.2: Study participant familarity with SQL, graphs, and the Cypher language in particular. Participants had a wide variety of skill levels with SQL, DBMSs, and graphs. Most particpants had little-to-no experience with the Cypher querying language.

from a wide variety of skill levels. Surprisingly we found no statistically significant correlations (we had hypothesized a slight inverse correlation) between reported scores and time taken per query task. The closest was general familiarity with SQL and DBMSs; however, the confidence was not sufficient to claim significance.

REFERENCES

[1] A. L. Traud, E. D. Kelsic, P. J. Mucha, and M. A. Porter, "Comparing community structure to characteristics in online collegiate social networks," *SIAM review*, vol. 53, no. 3, pp. 526–543, 2011.

[2] Z. Yang, J. Peltonen, and S. Kaski, "Optimization equivalence of divergences improves neighbor embedding.," in *ICML*, 2014, pp. 460–468.

[3] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: An open source software for exploring and manipulating networks," 2009.

[4] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu, *Xquery 1.0: An xml query language*, 2002.

[5] J. Clark, S. DeRose, *et al.*, *Xml path language (xpath) version 1.0*, 1999.

[6] J. Pérez, M. Arenas, and C. Gutierrez, "Semantics and complexity of sparql," in *International semantic web conference*, Springer, 2006, pp. 30–43.

[7] D. Montag, "Understanding neo4j scalability," Neo Technology, Tech. Rep., Jan. 2013.

[8] T. Lau and E. Horvitz, "Patterns of search: Analyzing and modeling web query refinement," in *UM99 User Modeling*, Springer, 1999, pp. 119–128.

[9] B. Vélez, R. Weiss, M. A. Sheldon, and D. K. Gifford, "Fast and effective query refinement," in *ACM SIGIR Forum*, ACM, vol. 31, 1997, pp. 6–15.

[10] R. Pienta, J. Abello, M. Kahng, and D. H. Chau, "Scalable graph exploration and visualization: Sensemaking challenges and opportunities," in *2015 International Conference on Big Data and Smart Computing, BIGCOMP 2015, Jeju, South Korea, February 9-11, 2015*, 2015, pp. 271–278.

[11] I. Herman, G. Melançon, and M. S. Marshall, "Graph visualization and navigation in information visualization: A survey," *IEEE TVCG*, vol. 6, no. 1, pp. 24–43, 2000.

[12] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. van Wijk, J.-D. Fekete, and D. Fellner, "Visual analysis of large graphs: State-of-the-art and future research challenges," *Computer Graphics Forum*, vol. 30, no. 6, pp. 1719–1749, 2011.

[13] F. Beck, M. Burch, S. Diehl, and D. Weiskopf, "The state of the art in visualizing dynamic graphs," in *EuroVis - STARs*, Eurographics Association, 2014, pp. 83–103.

[14] J. Leskovec and C. Faloutsos, "Sampling from large graphs," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '06, Philadelphia, PA, USA: ACM, 2006, pp. 631–636, ISBN: 1-59593-339-5.

[15] N. Elmqvist, T. N. Do, H. Goodell, N. Henry, and J. D. Fekete, "Zame: Interactive large-scale graph visualization," in *2008 IEEE Pacific Visualization Symposium*, Mar. 2008, pp. 215–222.

[16] V. Batagelj, W. Didimo, G. Liotta, P. Palladino, and M. Patrignani, "Visual analysis of large graphs using (x,y)-clustering and hybrid visualizations," in *2010 IEEE Pacific Visualization Symposium (PacificVis)*, Mar. 2010, pp. 209–216.

[17] G. D. Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Graph drawing: algorithms for the visualization of graphs*. Prentice Hall PTR, 1998.

[18] G. W. Furnas, "Generalized fisheye views," *SIGCHI Bull.*, vol. 17, no. 4, pp. 16–23, Apr. 1986.

[19] F. Van Ham and A. Perer, ""search, show context, expand on demand": Supporting large graph exploration with degree-of-interest," *IEEE TVCG*, vol. 15, no. 6, pp. 953–960, 2009.

[20] D. H. Chau, A. Kittur, J. I. Hong, and C. Faloutsos, "Apolo: Interactive large graph sensemaking by combining machine learning and visualization," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2011, pp. 739–742.

[21] M. Bilenko and R. W. White, "Mining the search trails of surfing crowds: Identifying relevant websites from user activity," in *Proceedings of WWW*, ser. WWW '08, Beijing, China: ACM, 2008, pp. 51–60, ISBN: 978-1-60558-085-2.

[22] A. Singla, R. White, and J. Huang, "Studying trailfinding algorithms for enhanced web search," in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '10, ACM, 2010, pp. 443–450, ISBN: 978-1-4503-0153-4.

[23] R. W. White and J. Huang, "Assessing the scenic route: Measuring the value of search trails in web logs," in *SIGIR*, ser. SIGIR '10, Geneva, Switzerland: ACM, 2010, pp. 587–594, ISBN: 978-1-4503-0153-4.

[24] J. Teevan, C. Alvarado, M. S. Ackerman, and D. R. Karger, "The perfect search engine is not enough: A study of orienteering behavior in directed search," in *Proceedings of CHI*, ser. CHI '04, Vienna, Austria: ACM, 2004, pp. 415–422, ISBN: 1-58113-702-8.

[25] R. West and J. Leskovec, "Human wayfinding in information networks," in *Proceedings of the 21st international conference on World Wide Web*, ACM, 2012, pp. 619–628.

[26] T. Wang, Y. Chen, Z. Zhang, T. Xu, L. Jin, P. Hui, B. Deng, and X. Li, "Understanding graph sampling algorithms for social network analysis," in *2011 31st International Conference on Distributed Computing Systems Workshops*, Jun. 2011, pp. 123–128.

[27] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *Proceedings of the 4th ACM European Conference on Computer Systems*, ser. EuroSys '09, Nuremberg, Germany: ACM, 2009, pp. 205–218, ISBN: 978-1-60558-482-9.

[28] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou, "Walking in facebook: A case study of unbiased sampling of osns," in *INFOCOM, 2010 Proceedings IEEE*, Mar. 2010, pp. 1–9.

[29] B. Ribeiro and D. Towsley, "Estimating and sampling graphs with multidimensional random walks," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10, Melbourne, Australia: ACM, 2010, pp. 390–403, ISBN: 978-1-4503-0483-2.

[30] V. Krishnamurthy, M. Faloutsos, M. Chrobak, L. Lao, J. H. Cui, and A. G. Percus, "Reducing large internet topologies for faster simulations," *NETWORKING'05*, pp. 328–341, 2005.

[31] P. Hu and W. C. Lau, "A survey and taxonomy of graph sampling," *CoRR*, vol. abs/1308.5865, 2013.

[32] S. Lee, P.-J. Kim, and H. Jeong, "Statistical properties of sampled networks," *Phys. Rev. E*, vol. 73, p. 016 102, 1 Jan. 2006.

[33] Y. Jia, J. Hoberock, M. Garland, and J. Hart, "On the visualization of social and other scale-free networks," *IEEE TVCG*, vol. 14, no. 6, pp. 1285–1292, Nov. 2008.

[34] S. Chen, F. Cerda, P. Rizzo, J. Bielak, J. H. Garrett, and J. Kovačević, "Semi-supervised multiresolution classification using adaptive graph filtering with application to indirect bridge structural health monitoring," *IEEE Transactions on Signal Processing*, vol. 62, no. 11, pp. 2879–2893, 2014.

[35] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE TVCG*, vol. 12, no. 5, pp. 741–748, Sep. 2006.

[36] D. Holten and J. J. Van Wijk, "Force-directed edge bundling for graph visualization," in *Computer graphics forum*, Wiley Online Library, vol. 28, 2009, pp. 983–990.

[37] E. R. Gansner, Y. Hu, S. North, and C. Scheidegger, "Multilevel agglomerative edge bundling for visualizing large graphs," in *2011 IEEE Pacific Visualization Symposium*, IEEE, 2011, pp. 187–194.

[38] O. Ersoy, C. Hurter, F. Paulovich, G. Cantareiro, and A. Telea, "Skeleton-based edge bundling for graph visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 12, pp. 2364–2373, 2011.

[39] G. Karypis and V. Kumar, "Metis-unstructured graph partitioning and sparse matrix ordering system, version 2.0," University of Minnesota, Department of Computer Science, Tech. Rep., 1995.

[40] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, "Graphcut textures: Image and video synthesis using graph cuts," *ACM Transactions on Graphics (ToG)*, vol. 22, no. 3, pp. 277–286, 2003.

[41] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01, San Francisco, California: ACM, 2001, pp. 269–274, ISBN: 1-58113-391-X.

[42] G. Karypis and V. Kumar, "Multilevelk-way partitioning scheme for irregular graphs," *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 96 –129, 1998.

[43] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29–123, 2009.

[44] G. Slota, K. Madduri, and S. Rajamanickam, "Pulp: Scalable multi-objective multi-constraint partitioning for small-world network," in *In the Proceedings of the 2014 IEEE Conference on Big Data*, IEEE, 2014.

[45] Y. Tian, R. A. Hankins, and J. M. Patel, "Efficient aggregation for graph summarization," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, ACM, 2008, pp. 567–580.

[46] J. Neville, M. Adler, and D. D. Jensen, "Clustering relational data using attribute and link information," in *Proceedings of the Workshop on Text Mining and Link Analysis, Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, 2003.

[47] M. Wattenberg, "Visual exploration of multivariate graphs," in *Proceedings of CHI*, ACM, 2006, pp. 811–819.

[48] Y. Zhou, H. Cheng, and J. X. Yu, "Graph clustering based on structural/attribute similarities," *Proc. VLDB Endow.*, vol. 2, no. 1, pp. 718–729, Aug. 2009.

[49] B. Shneiderman and A. Aris, "Network visualization by semantic substrates," *IEEE TVCG*, vol. 12, no. 5, pp. 733–740, 2006.

[50] G. M. Kuper and M. Y. Vardi, "A new approach to database logic," in *Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, ser. PODS '84, Waterloo, Ontario, Canada: ACM, 1984, pp. 86–96, ISBN: 0-89791-128-8.

[51] D. W. Shipman, "The functional data model and the data language daplex," in *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '79, Boston, Massachusetts: ACM, 1979, pp. 59–59, ISBN: 0-89791-001-X.

[52] N. Kiesel, A. Schuerr, and B. Westfechtel, "Gras, a graph oriented (software) engineering database system," *Inf. Syst.*, vol. 20, no. 1, pp. 21–51, Mar. 1995.

[53] H. S. Kunii, "Dbms with graph data model for knowledge handling," in *Proceedings of the 1987 Fall Joint Computer Conference on Exploring Technology: Today and Tomorrow*, ser. ACM '87, Dallas, Texas, USA: IEEE Computer Society Press, 1987, pp. 138–142, ISBN: 0-8186-0811-0.

[54] B. Amann and M. Scholl, "Gram: A graph data model and query languages," in *Proceedings of the ACM Conference on Hypertext*, ser. ECHT '92, Milan, Italy: ACM, 1992, pp. 201–211, ISBN: 0-89791-547-X.

[55] R. H. Güting, "Graphdb: Modeling and querying graphs in databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 297–308, ISBN: 1-55860-153-8.

[56] J. Han, H. E, G. Le, and J. Du, "Survey on nosql database," in *2011 6th International Conference on Pervasive Computing and Applications*, Oct. 2011, pp. 363–366.

[57] G. V. Paolini and G. R. D. Brozolo, "Data structures to vectorize cg algorithms for general sparsity patterns," *BIT Numerical Mathematics*, vol. 29, no. 4, pp. 703–718, 1989.

[58] M. M. Zloof, "Query-by-example: A data base language," *IBM Systems Journal*, vol. 16, no. 4, pp. 324–343, 1977.

[59] H.-J. Kim, H. F. Korth, and A. Silberschatz, "Picasso: A graphical query language," *Softw. Pract. Exper.*, vol. 18, no. 3, pp. 169–203, Mar. 1988.

[60] C. Ahlberg, C. Williamson, and B. Shneiderman, "Dynamic queries for information exploration: An implementation and evaluation," in *Proceedings of Conference on Human Factors in Computing Systems*, 1992, pp. 619–626.

[61] B. Shneiderman, "Dynamic queries for visual information seeking," *IEEE Software*, vol. 11, no. 6, pp. 70–77, 1994.

[62] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini, "Visual query systems for databases: A survey," *Journal of Visual Languages & Computing*, vol. 8, no. 2, pp. 215–260, 1997.

[63] S. Ceri, S. Comai, P. Fraternali, S. Paraboschi, L. Tanca, and E. Damiani, "Xml-gl: A graphical language for querying and restructuring xml documents," in *Proceedings of the Italian Symposium on Advanced Database Systems (SEBD)*, 1999, pp. 151–165.

[64] W. Ni and T. W. Ling, "Glass: A graphical query language for semi-structured data," in *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*, 2003, pp. 363–370.

[65] F. Hogenboom, V. Milea, F. Frasincar, and U. Kaymak, "Rdf-gl: A sparql-based graphical query language for rdf," in *Emergent Web Intelligence: Advanced Information Retrieval*, ser. Advanced Information and Knowledge Processing, Springer London, 2010, pp. 87–116.

[66] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, ser. STOC '71, Shaker Heights, Ohio, USA: ACM, 1971, pp. 151–158.

[67] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, pp. 31–42, Jan. 1976.

[68] B. T. Messmer and H. Bunke, "Subgraph isomorphism detection in polynomial time on preprocessed model graphs," in *Recent Developments in Computer Vision: Second Asian Conference on Computer Vision, ACCV '95 Singapore, December*

5–8, 1995 Invited Session Papers. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 373–382, ISBN: 978-3-540-49448-5.

[69] B. Luo and E. R. Hancock, "Structural graph matching using the em algorithm and singular value decomposition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 10, pp. 1120–1136, 2001.

[70] S. Umeyama, "An eigendecomposition approach to weighted graph matching problems," *IEEE transactions on pattern analysis and machine intelligence*, vol. 10, no. 5, pp. 695–703, 1988.

[71] A. D. Cross, R. C. Wilson, and E. R. Hancock, "Inexact graph matching using genetic search," *Pattern Recognition*, vol. 30, no. 6, pp. 953–970, 1997.

[72] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *International journal of pattern recognition and artificial intelligence*, vol. 18, no. 03, pp. 265–298, 2004.

[73] B. Aleman-Meza, C. Halaschek-Wiener, S. S. Sahoo, A. Sheth, and I. B. Arpinar, "Template based semantic similarity for security applications," in *Intelligence and Security Informatics: IEEE International Conference on Intelligence and Security Informatics, ISI 2005, Atlanta, GA, USA, May 19-20, 2005. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 621–622, ISBN: 978-3-540-32063-0.

[74] T. Coffman, S. Greenblatt, and S. Marcus, "Graph-based technologies for intelligence analysis," *Commun. ACM*, vol. 47, no. 3, pp. 45–47, Mar. 2004.

[75] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad, "Fast best-effort pattern matching in large attributed graphs," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '07, San Jose, California, USA: ACM, 2007, pp. 737–746, ISBN: 978-1-59593-609-7.

[76] R. Pienta, A. Tamersoy, H. Tong, and D. H. Chau, "MAGE: matching approximate patterns in richly-attributed graphs," in *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*, 2014, pp. 585–590.

[77] 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*, IEEE, 2008, pp. 963–966.

[78] A. Khan, Y. Wu, C. C. Aggarwal, and X. Yan, "Nema: Fast graph search with label similarity," *PVLDB*, vol. 6, no. 3, 2013.

[79] Y. Tian and J. Patel, "Tale: A tool for approximate large graph matching," in *ICDE*, 2008.

[80] W. Fan, X. Wang, and Y. Wu, "Diversified top-k graph pattern matching," *PVLDB*, vol. 6, no. 13, 2013.

[81] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, and Y. Wu, "Incremental graph pattern matching," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '11, Athens, Greece: ACM, 2011, pp. 925–936, ISBN: 978-1-4503-0661-4.

[82] 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.

[83] N. Cao, Y.-R. Lin, L. Li, and H. Tong, "G-miner: Interactive visual group mining on multivariate graphs," in *Proc. CHI*, ACM, 2015, pp. 279–288.

[84] 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*, ACM, 2007, pp. 201–210.

[85] A. Tamersoy, B. Xie, S. L. Lenkey, B. R. Routledge, D. H. Chau, and S. B. Navathe, "Inside insider trading: Patterns & discoveries from a large scale exploratory analysis," in *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ACM, 2013, pp. 797–804.

[86] F. Schreiber and H. Schwöbbermeyer, "Mavisto: A tool for the exploration of network motifs," *Bioinformatics*, vol. 21, no. 17, pp. 3572–3574, 2005.

[87] E Ted, H. G. Goldberg, A. Memory, W. T. Young, B. Rees, R. Pierce, D. Huang, M. Reardon, D. A. Bader, E. Chow, *et al.*, "Detecting insider threats in a real corporate database of computer usage activity," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2013, pp. 1393–1401.

[88] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon, "Network motifs in the transcriptional regulation network of escherichia coli," *Nature Genetics*, vol. 31, no. 1, pp. 64–68, 2002.

[89] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon, "Network motifs: Simple building blocks of complex networks.," *Science*, vol. 298, no. 5594, pp. 824–827, Oct. 2002.

[90] J. A. Grochow and M. Kellis, "Network motif discovery using subgraph enumeration and symmetry-breaking," in *Proceedings of the 11th Annual International Conference on Research in Computational Molecular Biology*, ser. RECOMB'07, Oakland, CA, USA: Springer-Verlag, 2007, pp. 92–106, ISBN: 978-3-540-71680-8.

[91] X. Yan and J. Han, "Gspan: Graph-based substructure pattern mining," in *ICDM*, 2002, pp. 721–724.

[92] D. Koutra, U Kang, J. Vreeken, and C. Faloutsos, "Vog: Summarizing and understanding large graphs," in *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2014.

[93] C. Dunne and B. Shneiderman, "Motif simplification: Improving network visualization readability with fan, connector, and clique glyphs," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '13, Paris, France: ACM, 2013, pp. 3247–3256, ISBN: 978-1-4503-1899-0.

[94] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer, "Learning probabilistic relational models," in *Relational data mining*, Springer, 2001, pp. 307–335.

[95] W. Wang, H. Jiang, H. Lu, and J. X. Yu, "Bloom histogram: Path selectivity estimation for xml data with updates," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, ser. VLDB '04, Toronto, Canada: VLDB Endowment, 2004, pp. 240–251, ISBN: 0-12-088469-0.

[96] L. Getoor and L. Mihalkova, "Learning statistical models from relational data," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '11, Athens, Greece: ACM, 2011, pp. 1195–1198, ISBN: 978-1-4503-0661-4.

[97] A. Chapman and H. V. Jagadish, "Why not?" In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09, Providence, Rhode Island, USA: ACM, 2009, pp. 523–534, ISBN: 978-1-60558-551-2.

[98] S. Roy and D. Suciu, "A formal approach to finding explanations for database queries," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14, Snowbird, Utah, USA: ACM, 2014, pp. 1579–1590, ISBN: 978-1-4503-2376-5.

[99] N. Shervashidze, P. Schweitzer, E. J. v. Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. Sep, pp. 2539–2561, 2011.

[100] R. C. Bunescu and R. J. Mooney, "A shortest path dependency kernel for relation extraction," in *Proceedings of the conference on human language technology and empirical methods in natural language processing*, Association for Computational Linguistics, 2005, pp. 724–731.

[101] X. Jiang, A. Munger, and H. Bunke, "An median graphs: Properties, algorithms, and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 10, pp. 1144–1151, 2001.

[102] G. H. Bakır, A. Zien, and K. Tsuda, "Learning to find graph pre-images," in *Joint Pattern Recognition Symposium*, Springer, 2004, pp. 253–261.

[103] H. Kashima, K. Tsuda, and A. Inokuchi, "Marginalized kernels between labeled graphs," in *ICML*, vol. 3, 2003, pp. 321–328.

[104] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt, "Graph kernels," *J. Mach. Learn. Res.*, vol. 11, pp. 1201–1242, Aug. 2010.

[105] L. A. Zager and G. C. Verghese, "Graph similarity scoring and matching," *Applied Mathematics Letters*, vol. 21, no. 1, pp. 86 –94, 2008.

[106] M. Berlingerio, D. Koutra, T. Eliassi-Rad, and C. Faloutsos, "Netsimile: A scalable approach to size-independent network similarity," *CoRR*, vol. abs/1209.2684, 2012.

[107] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, 2016, pp. 855–864.

[108] A. Narayanan, M. Chandramohan, L. Chen, Y. Liu, and S. Saminathan, "Subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs," *arXiv preprint arXiv:1606.08928*, 2016.

[109] S. van den Elzen, D. Holten, J. Blaas, and J. J. van Wijk, "Reducing snapshots to points: A visual analytics approach to dynamic network exploration," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 1–10, Jan. 2016.

[110] H. Hotelling, "Analysis of a complex of statistical variables into principal components.," *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[111] K. P. F.R.S., "Liii. on lines and planes of closest fit to systems of points in space," *Philosophical Magazine Series 6*, vol. 2, no. 11, pp. 559–572, 1901.

[112] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel principal component analysis," in *International Conference on Artificial Neural Networks*, Springer, 1997, pp. 583–588.

[113] S. Mika, B. Schölkopf, A. J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch, "Kernel pca and de-noising in feature spaces.," in *NIPS*, vol. 11, 1998, pp. 536–542.

[114] J. B. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[115] ——, "Nonmetric multidimensional scaling: A numerical method," *Psychometrika*, vol. 29, no. 2, pp. 115–129, 1964.

[116] L. van der Maaten and G. E. Hinton, "Visualizing high-dimensional data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[117] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[118] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011.

[119] V. Rokhlin, A. Szlam, and M. Tygert, "A randomized algorithm for principal component analysis," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 3, pp. 1100–1124, 2009.

[120] I. Borg and P. J. Groenen, *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[121] L. Van Der Maaten, "Accelerating t-sne using tree-based algorithms.," *Journal of machine learning research*, vol. 15, no. 1, pp. 3221–3245, 2014.

[122] D. L. Donoho and C. Grimes, "Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data," *Proceedings of the National Academy of Sciences*, vol. 100, no. 10, pp. 5591–5596, 2003.

[123] S. M. Emran and N. Ye, "Robustness of chi-square and canberra distance metrics for computer intrusion detection," *Quality and Reliability Engineering International*, vol. 18, no. 1, pp. 19–28, 2002.

[124] G. Kossinets and D. J. Watts, "Empirical analysis of an evolving social network," *science*, vol. 311, no. 5757, pp. 88–90, 2006.

[125] S. Aliakbary, J. Habibi, and A. Movaghar, "Feature extraction from degree distribution for comparison and analysis of complex networks," *The Computer Journal*, bxv007, 2015.

[126] R. Amar, J. Eagan, and J. Stasko, "Low-level components of analytic activity in information visualization," in *Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, ser. INFOVIS '05, Washington, DC, USA: IEEE Computer Society, 2005, pp. 15–, ISBN: 0-7803-9464-x.

[127] J. S. Yi, Y. A. KANG, J. STASKO, and J JACKO, "Gspan: Graph-based substructure pattern mining," in *ICDM*, 2002, pp. 721–724.

[128] H. Kang, C. Plaisant, B. Lee, and B. B. Bederson, "Netlens: Iterative exploration of content-actor network data," in *2006 IEEE Symposium On Visual Analytics Science And Technology*, Oct. 2006, pp. 91–98.

[129] B. Lee, C. S. Parr, C. Plaisant, B. B. Bederson, V. D. Veksler, W. D. Gray, and C. Kotfila, "Treeplus: Interactive exploration of networks with enhanced tree layouts," *IEEE TVCG*, vol. 12, no. 6, pp. 1414–1426, 2006.

[130] T. Moscovich, F. Chevalier, N. Henry, E. Pietriga, and J.-D. Fekete, "Topology-aware navigation in large networks," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09, Boston, MA, USA: ACM, 2009, pp. 2319–2328, ISBN: 978-1-60558-246-7.

[131] B. Lee, C. Plaisant, C. S. Parr, J.-D. Fekete, and N. Henry, "Task taxonomy for graph visualization," in *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*, ACM, 2006, pp. 1–5.

[132] S. K. Card, J. D. Mackinlay, and B. Shneiderman, *Readings in information visualization: using vision to think*. Morgan Kaufmann, 1999.

[133] C. Ware, *Information Visualization: Perception for Design*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, ISBN: 1-55860-511-8.

[134] M. J. McGuffin and I. Jurisica, "Interaction techniques for selecting and manipulating subgraphs in network visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 937–944, Nov. 2009.

[135] J. Abello and F. van Ham, "Matrix zoom: A visual interface to semi-external graphs," in *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, 2004, pp. 183–190.

[136] N. Elmqvist, P. Dragicevic, and J. D. Fekete, "Color lens: Adaptive color scale optimization for visual exploration," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 6, pp. 795–807, Jun. 2011.

[137] N. Henry, J. D. Fekete, and M. J. McGuffin, "Nodetrix: A hybrid visualization of social networks," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1302–1309, Nov. 2007.

[138] S. Zhao, M. J. McGuffin, and M. H. Chignell, "Elastic hierarchies: Combining treemaps and node-link diagrams," in *IEEE Symposium on Information Visualization, 2005. INFOVIS 2005.*, Oct. 2005, pp. 57–64.

[139] C. Tominski, J. Abello, F. van Ham, and H. Schumann, "Fisheye tree views and lenses for graph visualization," in *Information Visualization, 2006. IV 2006. Tenth International Conference on*, Jul. 2006, pp. 17–24.

[140] S. van den Elzen and J. J. van Wijk, "Multivariate network exploration and presentation: From detail to overview via selections and aggregations," *IEEE TVCG*, vol. 20, no. 12, pp. 2310–2319, 2014.

[141] C. Tominski, J. Abello, and H. Schumann, "Technical section: Cgv-an interactive graph visualization system," *Comput. Graph.*, vol. 33, no. 6, pp. 660–678, Dec. 2009.

[142] E. Bertini, M. Rigamonti, and D. Lalanne, "Extended excentric labeling," in *Proceedings of the 11th Eurographics / IEEE - VGTC Conference on Visualization*, ser. EuroVis'09, Berlin, Germany, 2009, pp. 927–934.

[143] J.-D. Fekete and C. Plaisant, "Excentric labeling: Dynamic neighborhood labeling for data visualization," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '99, Pittsburgh, Pennsylvania, USA: ACM, 1999, pp. 512–519, ISBN: 0-201-48559-1.

[144] J. Heer and D. Boyd, "Vizster: Visualizing online social networks," in *Proceedings of the Proceedings of the 2005 IEEE Symposium on Information Visualization*, ser. INFOVIS '05, IEEE Computer Society, 2005, pp. 5–, ISBN: 0-7803-9464-x.

[145] S. K. Card and D. Nation, "Degree-of-interest trees: A component of an attention-reactive user interface," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, ser. AVI '02, Trento, Italy: ACM, 2002, pp. 231–245, ISBN: 1-58113-537-8.

[146] J. Heer and S. K. Card, "Doitrees revisited: Scalable, space-constrained visualization of hierarchical data," in *Proceedings of the Working Conference on Advanced*

Visual Interfaces, ser. AVI '04, Gallipoli, Italy: ACM, 2004, pp. 421–424, ISBN: 1-58113-867-9.

[147] J. Abello, S. Hadlak, H. Schumann, and H. Schulz, "A modular degree-of-interest specification for the visual analysis of large dynamic networks," *IEEE TVCG*, vol. 20, no. 3, pp. 337–350, 2014.

[148] A. Lex, C. Partl, D. Kalkofen, M. Streit, S. Gratzl, A. M. Wassermann, D. Schmalstieg, and H. Pfister, "Entourage: Visualizing relationships between biological pathways using contextual subsets," *IEEE TVCG*, vol. 19, no. 12, pp. 2536–2545, Dec. 2013.

[149] R. Pienta, M. B. Kahng, Z. Lin, J. Vreeken, P. Talukdar, J. Abello, G. Parameswaran, and D. H. P. Chau., "Facets: Adaptive local exploration of large graphs," in *SIAM International Conference on Data Mining (SDM) 2017*, 2017.

[150] R. Pienta, Z. Lin, M. Kahng, J. Vreeken, P. P. Talukdar, J. Abello, G. Parameswaran, and D. H. Chau., "Adaptivenav: Adaptive discovery of interesting and surprising nodes in large graphs," in *IEEE VIS*, 2015.

[151] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *VL*, IEEE, 1996, pp. 336–343.

[152] D. A. Keim, "Visual exploration of large data sets," *ACM Communications*, vol. 44, no. 8, pp. 38–44, 2001.

[153] C. Plaisant, B. Milash, A. Rose, S. Widoff, and B. Shneiderman, "Lifelines: Visualizing personal histories," in *SIGCHI*, ACM, 1996, pp. 221–227.

[154] D. A. Keim, "Information visualization and visual data mining," *IEEE TVCG*, vol. 8, no. 1, pp. 1–8, 2002.

[155] S. Kairam, N. H. Riche, S. M. Drucker, R. Fernandez, and J. Heer, "Refinery: Visual exploration of large, heterogeneous networks through associative browsing," *Comput. Graph. Forum*, vol. 34, no. 3, pp. 301–310, 2015.

[156] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," 4, ACM, vol. 29, 1999, pp. 251–262.

[157] L. Akoglu, M. McGlohon, and C. Faloutsos, "Oddball: Spotting anomalies in weighted graphs," in *Advances in Knowledge Discovery and Data Mining*, Springer, 2010, pp. 410–421.

[158] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Stanford, Technical Report SIDL-WP-1999-0120 1999-66, Nov. 1999.

[159] T. H. Haveliwala, "Topic-sensitive pagerank," in *WWW02*, Honolulu, Hawaii, USA: ACM, 2002, pp. 517–526, ISBN: 1-58113-449-5.

[160] H. Tong, C. Faloutsos, and J.-Y. Pan, "Fast random walk with restart and its applications," in *ICDM06*, Washington, DC, USA: IEEE, 2006, pp. 613–622, ISBN: 0-7695-2701-9.

[161] K. Onuma, H. Tong, and C. Faloutsos, "Tangent: A novel,'surprise me', recommendation algorithm," in *KDD09*, ACM, 2009, pp. 657–666.

[162] P. Kontkanen and P. Myllymäki, "MDL histogram density estimation," 2007.

[163] T. Lee, Z. Wang, H. Wang, and S. won Hwang, "Attribute extraction and scoring: A probabilistic approach," in *ICDE13*, 2013, pp. 194–205.

[164] M. Dork, N. H. Riche, G. Ramos, and S. Dumais, "Pivotpaths: Strolling through faceted information spaces," *IEEE TVCG*, vol. 18, no. 12, pp. 2709–2718, 2012.

[165] F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens, "Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation," vol. 19, no. 3, pp. 355–369, Mar. 2007.

[166] F. Wang, S. Ma, L. Yang, and T. Li, "Recommendation on item graphs," in *ICDM06*, Dec. 2006, pp. 1119–1123.

[167] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *TOIS*, vol. 22, no. 1, pp. 5–53, 2004.

[168] J. Leskovec and A. Krevl, *SNAP Datasets: Stanford large network dataset collection*, `http://snap.stanford.edu/data`, Jun. 2014.

[169] M. Mongioví, R. D. Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan, "Sigma: A set-cover-based approach for inexact graph matching," *J. Bioinformatics and Computational Biology*, vol. 8, no. 2, 2010.

[170] J. Cheng, X. Zeng, and J. X. Yu, "Top-k graph pattern matching over large graphs," in *ICDE*, 2013, pp. 1033–1044.

[171] H. Whitney, "Congruent graphs and the connectivity of graphs," *AJAM*, vol. 54, no. 1, 1932.

[172] H. Tong and C. Faloutsos, "Center-piece subgraphs: Problem definition and fast solutions," in *KDD*, 2006.

[173] N. Z. Gong, A. Talwalkar, L. W. Mackey, L. Huang, E. C. R. Shin, E. Stefanov, E. Shi, and D. Song, "Predicting links and inferring attributes using a social-attribute network (san)," *CoRR*, 2011.

[174] N. Z. Gong, W. Xu, L. Huang, P. Mittal, E. Stefanov, V. Sekar, and D. Song, "Evolution of social-attribute networks: Measurements, modeling, and implications using google+," *CoRR*, 2012.

[175] P. Erdös and A. Rényi, "On random graphs, I," *Publicationes Mathematicae (Debrecen)*, vol. 6, pp. 290–297, 1959.

[176] D. J. Watts and S. H. Strogatz, "Collective dynamics of /'small-world/' networks," *Nature*, vol. 393, pp. 440–442, 1998.

[177] S. Guha, R. Rastogi, and K. Shim, "Rock: A robust clustering algorithm for categorical attributes," in *Data Engineering, 1999. Proceedings., 15th International Conference on*, Mar. 1999, pp. 512–521.

[178] 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*, vol. 4, no. 1, pp. 1–17, 2014.

[179] 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*, ser. AVI '16, Bari, Italy: ACM, 2016, pp. 272–279, ISBN: 978-1-4503-4131-8.

[180] R. Pienta, F. Hohman, A. Tamersoy, A. Endert, S. Navathe, H. Tong, and D. H. Chau, "Visual graph query construction and refinement [best demo: Honorable mention]," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17, Chicago, Illinois, USA: ACM, 2017, pp. 1587–1590, ISBN: 978-1-4503-4197-4.

[181] R. Pienta, A. Tamersoy, H. Tong, A. Endert, and D. H. P. Chau, "Interactive querying over large network data: Scalability, visualization, and interaction design," in *Proceedings of the 20th International Conference on Intelligent User Interfaces Companion, IUI 2015, Atlanta, GA, USA, March 29 - April 01, 2015*, 2015, pp. 61–64.

[182] 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.

[183] A. Hakeem, M. W. Lee, O. Javed, and N. Haering, "Semantic video search using natural language queries," in *Proc. Multimedia*, ACM, 2009, pp. 605–608.

[184] F. Holzschuher and R. Peinl, "Performance of graph query languages: Comparison of cypher, gremlin and native access in neo4j," in *Proc. Joint EDBT/ICDT Workshops*, ACM, 2013, pp. 195–204.

[185] K. Kaur and R. Rani, "Modeling and querying data in nosql databases," in *Proc. Big Data*, IEEE, 2013.

[186] A. Jindal and S. Madden, "Graphiql: A graph intuitive query language for relational databases," in *Proc. Big Data*, IEEE, 2014, pp. 441–450.

[187] R. C. Littell, G. A. Milliken, W. W. Stroup, and R. D. Wolfinger, *SAS System for Mixed Models.* Cary, North Carolina: SAS Institute, Inc: SAS Institute Inc., 2006, p. 413.

[188] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, 1:1–1:39, Feb. 2008.

[189] R. Pienta, A. Endert, S. Navathe, and D. H. Chau, "Making sense of graph query results: Interactive summarization and exploration (poster)," in *IEEE Visual Analytics Science and Technology, VAST 2016*, 2016.

[190] D. M. Russell, M. J. Stefik, P. Pirolli, and S. K. Card, "The cost structure of sensemaking," in *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, ser. CHI '93, Amsterdam, The Netherlands: ACM, 1993, pp. 269–276, ISBN: 0-89791-575-5.

[191] K. J. Holyoak and P. Thagard, *Mental Leaps: Analogy in Creative Thought*. Cambridge, MA, USA: MIT Press, 1995, ISBN: 0-262-08233-0.

[192] P. Pirolli and S. Card, "The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis," in *Proceedings of international conference on intelligence analysis*, vol. 5, 2005, pp. 2–4.

[193] C. North and B. Shneiderman, "Snap-together visualization: A user interface for coordinating visualizations via relational schemata," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, ser. AVI '00, Palermo, Italy: ACM, 2000, pp. 128–135, ISBN: 1-58113-252-2.

[194] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, "Density-based clustering," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 231–240, 2011.

[195] J. Stahnke, M. Dörk, B. Müller, and A. Thom, "Probing projections: Interaction techniques for interpreting arrangements and errors of dimensionality reductions," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 629–638, Jan. 2016.

[196] B. Schölkopf and C. J. Burges, *Advances in kernel methods: support vector learning*. MIT press, 1999.

[197] G. N. Lance and W. T. Williams, "Computer programs for hierarchical polythetic classification ("similarity analyses")," *The Computer Journal*, vol. 9, no. 1, pp. 60–64, 1966.

[198] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu, "Density-based clustering in spatial databases: The algorithm gdbscan and its applications," *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 169–194, Jun. 1998.

[199] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[200] M. E. Newman, "Coauthorship networks and patterns of scientific collaboration," *Proceedings of the national academy of sciences*, vol. 101, no. suppl 1, pp. 5200–5205, 2004.

[201] D. Tunkelang, "Faceted search," *Synthesis lectures on information concepts, retrieval, and services*, vol. 1, no. 1, pp. 1–80, 2009.

[202] B. Zheng, W. Zhang, and X. F. B. Feng, "A survey of faceted search," *Journal of Web engineering*, vol. 12, no. 1&2, pp. 041–064, 2013.

[203] O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev, "Beyond basic faceted search," in *Proceedings of the 2008 International Conference on Web Search and Data Mining*, ACM, 2008, pp. 33–44.

[204] B. Kules, R. Capra, M. Banta, and T. Sierra, "What do exploratory searchers look at in a faceted search interface?" In *Proceedings of the 9th ACM/IEEE-CS joint conference on Digital libraries*, ACM, 2009, pp. 313–322.

[205] K. Oflazer and Y. Yilmaz, "Vi-xfst: A visual regular expression development environment for xerox finite state tool," in *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*, Association for Computational Linguistics, 2004, pp. 86–93.

[206] A Blackwell, "Swyn: A visual representation for regular expressions," *Your Wish Is My Command: Programming by Example*, pp. 245–270, 2001.

[207] R. Baumgartner, S. Flesca, and G. Gottlob, "Visual web information extraction with lixto," in *VLDB*, vol. 1, 2001, pp. 119–128.

[208] J. Heer and M. Bostock, "Crowdsourcing graphical perception: Using mechanical turk to assess visualization design," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2010, pp. 203–212.