A Visual Analytic Framework for Graphs

by
Anna Arpi Shaverdian

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2012

Doctoral Committee:

Professor Hosagrahar V. Jagadish, Chair
Professor George Michailidis
Assistant Professor Michael J. Cafarella
Assistant Professor Kristen R. LeFevre
Chief Scientist Pak Chung Wong, Pacific Northwest National Laboratory

TABLE OF CONTENTS

LIST OF FIGURES

Figure

CHAPTER I

Introduction

Visual Analytics is the science of analytical reasoning through visual interaction. In recent years there has been growing interest in this field as people recognize that a mix of human and machine intelligence can often be much more effective than either one alone.

Graphs are ubiquitous: high-throughput "omic" sciences use graphs to study pathways, computer networks use graphs to analyze communications, and almost everyone is involved in the explosive growth of online social networks. Graphs are also particularly amenable to visual representation. This dissertation focuses on graph data sets.

1.1 Motivation

While visual analytics holds out great promise, there remain many barriers to realizing this promise. We look at a few of them in the next subsection.

1.1.1 Lack of Systematic Methodology in Visual Analytics

It is no surprise that graph visual analytics has received a lot of attention as evidenced by the large number of tools and algorithms developed for this purpose.

As an effort to push the field forward by developing metrics for comparing different methods, the community holds an annual competition as part of the IEEE Visual Analytics Science and Technology (VAST) symposium. The VAST 2008 Challenge [46] presented

a fictitious political movement and asked participants to identify characteristic features of the criminal ring within the data set. The data set, which can be modeled as a graph, consisted of cell phone records over a 10 day period. The participants were given some clues about the ground truth embedded in the data sets and they were asked to identify the criminal ring.

This challenge resulted in over 50 entries. The judges were tasked to provide feedback based on: accuracy, process descriptions, analysis, and visualizations that were used to perform the analysis. Each entry provided a written analysis and five minute video describing their solution. Each entry had their own interpretation of the requirements for producing an analytical report. This variation resulted in a difficultly to systematically compare methods and tools. For example, is what tool A does to complete the task really different from tool Bs functionality? Moreover, how do we combine tool A and B?

The underlying cause for these problems is the lack of a systematic method to design graph tools. There is no common language that defines the basic graph data manipulation actions in visual analytics. We present a visual analytic graph algebra to meet this void.

1.1.2 Visual Analytics for Hairball Networks

As part of the effort to formalize the visual analytic science, researchers have identified a task taxonomy for graph analytics which includes identifying subgraphs, nodes, outliers, and communities in a network.

Solving these problems becomes very difficult, to nearly impossible for hairball networks. A hairball network is a network which looks like a hairball because there are too many nodes and edges at the current resolution for any interesting graph features to appear.

Given the multitude of sophisticated visualization tools to support analysis, there is a lack of utilizing indexing structures existing in graphs to aid in visual exploration tasks.

The theoretical graph community has identified several metrics to model complex graph

structures. Popular ones, such as degree and clustering coefficient, are widely used in graph software. These metrics can be used to build visual indexes and explore hairball networks. But the difficulty which remains is that an analyst might not know the mathematical meaning of the metric, nor how the metric relates to the graph's context.

Furthermore, these metrics change at increasing distances from the node, which adds more complexity into which metric would be most useful for the hairball network. For example, we can compare the topologies of the graphs induced at each node at increasing distances away from the node. In a hairball network summarizing a greater distance helps condense information and explore graph tasks.

We attack the problem of exploring graph tasks on hairball networks by presenting a tool built based on the visual analytic graph framework. We display graphs summarized with signatures based on multiple metrics at increasing distances and link these displays to the traditional force directed display. We show how this multi-modal exploration tool helps to solve graph tasks, such as identifying outliers, communities, and interesting trends.

1.1.3 Visual Analytics for Messy Networks and Complicated Workflows

Hairball networks present a complication to visual analytic tools. But another barrier to correct visual exploration is the inherent messy data in real world networks. Sometimes this complication means missing data in networks. Missing data is prevalent in high-throughput bioinformatics data sets because missing intrinsic attributes on nodes and edges occur when experimental results from different labs, with different procedures, are fused together. This missing information might affect how the analyst visualizes the data set. This dissertation presents a pattern-matching based algorithm to predict values for these missing attributes.

Another manifestation of messy real world networks is uncertainty. One popular concern is uncertainty on the graph elements. For example, we might not be sure of the exact

height of a person in a social network. But, the uncertainty this dissertation focuses on is query uncertainty. The 2009 VAST Challenge presents an example of query uncertainty. Similar to the previous years challenge, the participants were given a fictitious scenario and asked to identify the criminal ring structure in a network. The data set is a synthetic social network and participants were given two possible hypothesis structures to identify in the social network. In the possible structures, there existed uncertainty on the attribute values. For example, node A had $approximately$ 40 friends.

This uncertainty results in a situation where the analyst has an $idea$ of what she would like to search for in a graph, so an operation that produces exact results for a precisely stated query is too stringent. Further complicating this uncertainty is if there are multiple steps in the exploration process. The analyst starts with an initial uncertainty, and as she continues manipulating and visually exploring the data, the uncertainty is carried to the final results. Modeling and managing query uncertainty is another barrier to visual analytics. To address this issue, we introduce a probabilistic framework which incorporates uncertainty in the queries and provides a probabilistic assessment of the likelihood of the final obtained outcomes.

The above mentioned barriers to graph visual analytics all require an exploration component. An analyst starts with a raw graph data, which is possibly a hairball network or messy network, and she performs various operations during exploration. This workflow may be a composition of several operators, some of which may be very time consuming. For example, computing a metric which must traverse the entire graph is time consuming. Furthermore, the screen resolution and interactivity-time limits are other challenges to visual analytic tools.

We propose to work present optimization techniques given our visual analytic framework for graphs. Since these tools are interactive, the exploration needs to be fast enough

so that an analyst's work is not interrupted.

1.2 Key Contributions and Dissertation Outline

This dissertation introduces a visual analytic framework for graphs with the following components: an algebra, a multi-modal graph analysis tool, uncertain querying, and missing attribute interpolation. The visual analytic graph algebra, presented in Chapter III includes the following: (1) a formalized graph model, (2) an expressive predicate language, and (3) an algebra with associated operators.

The purpose of the operators is to manipulate the raw graph data before applying a visualization scheme. The selection operator zooms into a region of interest. The aggregation operator manipulates the data resolution with supernodes. And finally, the Visual Operator incorporates the interactive aspect of visual analytics in a work flow, which makes a visualization creation stage as flexible as the data manipulation stage.

The benefits of an algebra are now we can systematically replicate, compare, and assess graph visual analytics. Thus, a visual analytic algebra facilitates the production of graph information analysis. We present an example implementation of the algebra in Cytoscape, a popular bioinformatic graph exploration tool, case studies, and a usability study which studies the benefits of the algebra from visual exploration reproducibility. We demonstrate an example application of the algebra to analyze high throughput biological datasets. We reproduce statistical findings from previous genomic and metabolomic studies related to prostate cancer progression through visualizations created with the algebra.We also present optimization methods for repeated workflows to create a target visualization. We explore the visual analytic graph algebra rewrite rules which will result in fewer graph matching function calls.

Then we utilize this framework to study the other motivating problems listed above. In

Chapter IV we present GreenTrellis which is a mutli-modal graph exploratory tool. GreenTrellis displays multiple graph signature scatterplots to characterize the graph at different node localities. A graph signature captures a nodes local topology as a numeric vector that, when displayed on a low-dimensional scatterplot, reveals graph features. We present several rich signature representations based on different node characteristics. We demonstrate GreenTrellis multi-modal exploration through usability studies and procedures to solve various graph exploration tasks.

In Chapter V, we present a model for uncertain predicates in a network query exploration. This model guides the user from defining the uncertain query to presenting a range of results. The results are ranked according to the user's uncertainty. We build this model over the visual analytic framework and show how an uncertain query is carried over a composition of operators. We handle uncertain queries by expanding an uncertain query to a set of exact queries. The user places a probability on each exact query to represent how well it captures the uncertain query. In our framework we assume the graph remains exact and uncertainty only remains at the query level. We examine this framework through case studies.

In Chapter VI, we present a method to interpolate values for missing intrinsic node attributes on a graph. We treat the graph as the raw data for deducing similarities and supports for interpolation by performing a Bayesian inference analysis on interacting pairs. Then, we compare the query's interacting pairs with the information obtained to interpolate appropriate values. We demonstrate the effectiveness and broad applicability of our new interpolation approach using extensive experiments on real data sets. We explore graphs from a variety of applications including biological, trade, and social networks. These graphs have different network properties, attributes, and interactions. The effectiveness property we analyze is the precision of the results. The results show that our method is a

more effective interpolator compared to current state of the art methods.

We begin the dissertation with Related Work in Chapter II. Each chapter also includes related work specific to the material presented.

1.2.1 Motivating Overview Example

The framework consists of a collection of methodologies and tools which cohesively work to improve graph visual analytics. We use the VAST 2008 cell phone network challenge as a graph visual task which requires each of the components for effective exploration. The data set includes over 9000 calls records from 400 unique cell phones over a ten-day period. The analysis challenge is to identify and characterize a particular subnetwork. The subnetwork consists of five individuals. One of the individuals $might$ be identifier 200, who may speak with another individual very frequently. These two also speak with another individual in the group who coordinates high-level activities and communications within the network. This analysis must overcome difficulties in: data management and transformation, visual exploration, and uncertainty and missing information. To show how the components in the framework work to solve this challenge, we will revisit this overview example in each chapter.

CHAPTER II

Related Work

2.1 Overview

In this chapter, we review literature which overlaps related work for the chapters in this thesis. Each chapter also includes related work specific to the material presented.

2.2 Graph Layout and Visualization

The graph drawing community has studied drawing and layout issues for decades [25]. Di Battista et al. [9] and Sugiyama [51] review most major graph drawing algorithms and their applications. Card et al. [13] and Chen [14] summarize the major research in graph visualization. Also, the IEEE Symposium on Information Visualization [32] annually presents new research on graph visualization. Other notable multiple matrix metric visualization tool for graphs include GraphDice [10] and Viau et al. work in [54] which both present a hybrid multidimensional visualization method for network exploration. In [54], they use multiple scatterplot matrices [28] based on different network metrics and link these to a standard graph drawing. Their work shows ways to order and compare each of the scatterplots using parallel coordinates and other views. Our work differs from these tools in that we focus more on the graph signature designs and provide analytical schemes to use this visualization to solve real-world graph tasks. In GreenTrellis, we use the graph signatures to provide zoomed-in views of data, creating a coordinated signature

visualization for discovery exploration.

Besides graph metrics, there are several tools which visualize other graph features, such as intrinsic attributes, on scatterplots or histograms to understand graph structures. For example, ManyNets [22] uses histograms rather than scatterplots to compare network metrics for multiple networks simultaneously. GreenTrellis displays metrics over varying distances from nodes; therefore, network to network comparison is also possible by visualizing scatterplots at the distance of the graphs diameter. However, ManyNets histogram techniques are quite useful for several graphs simultaneously, while GreenTrellis is more suitable to study a single graph. In [55] a PivotGraph is used to summarize multi-variate graph information, which is a useful way to compare two attributes. However, a matrix of signatures used in GreenTrellis becomes more useful when the number of features exceeds two. In [44] nodes are organized according to their attributes. This work is orthogonal to GreenTrellis because we focus on attributes computed from relationships between nodes. In [48] tables of nodes and edges are exposed to analysts as spreadsheet pages. At this point, GreenTrellis does not support attribute value manipulation since we focus on visualizing topological features.

There are several noteworthy visualization tools for data analysis. For example, Cytoscape [15] is primarily designed for exploring biological networks. With a collection of user-community created plug-in tools, Cytoscape is able to do graph querying on basic attribute values and creation of new graphs based on selections. Other popular graphical analysis tools include Pajek [8] and Guess [2]. Pajek provides several predefined metrics for nodes and edges. This functionality is similar to our composition functions. Guess has a built in query language into its graph visualization tool. Tableau [56] uses a structured query language for data visualization of relational databases, cubes, and spreadsheets. ZAME [20] is a visualization tool for exploring graphs at a scale of millions of nodes and

edges. As pointed out above, our algebra incorporates the functionality necessary to manipulate, reduce, and analyze the graph. Such operations can directly communicate with other visualization tools through visualization operators to facilitate analysis of networks. Our visualization also uses brushing and linked views, which are explored by [50].

2.3 Graph Analysis

Several tools and methods have been created for analyzing graph data sets. One method is to capture the local topology of a graph as a signature to aid exploration of complex networks [60]. Other works use semantic and structural abstractions to analyze social networks [44], [31]. In regards to interactive graph analysis, [5] presents ways to select subgraphs and manipulate the graph to find interesting relationships. In [59] a breadth first search subgraph of a selected node is set into motion to improve interactive graph analysis. There are also systems, such as [58], [4], [1], that investigate different layouts to display complex networks, their nodes, edges, and attributes effectively for analysis. A key point to observe is that while there are several visual analytic tools for graph analysis, there is not a formal foundation to develop these systems.

Often to make sense of raw data, nodes are aggregated into supernodes and edges [61] to visualize new patterns. For example, [3] maintain hierarchical structure of aggregated nodes. We provide an aggregation operator which maintains hierarchy of the aggregated nodes through descriptive attributes and edges to the original structure. The user decides which visualization scheme to apply to this manipulated data. At the data level, no information is lost to an aggregated supernode.

CHAPTER III

Algebra

3.1 Introduction

Visual Analytics is the science of analytical reasoning through visual interaction. In recent years there has been growing interest in this field as people recognize that a mix of human and machine intelligence can often be much more effective than either one alone.

Graphs are ubiquitous: high-throughput "omic" sciences use graphs to study pathways, computer networks use graphs to analyze communications, and almost everyone is involved in the explosive growth of online social networks. Graphs are also particularly amenable to visual representation.

It is no surprise that graph visual analytics has received a lot of attention as evidenced by the large number of tools and algorithms developed for this purpose. Using these tools and algorithms, there is also a growing body of literature describing both visual analytics systems as well as problems successfully addressed through visual analytics. These systems can be very informative, but usually constrain the reader in realizing their full value for several reasons, listed below.

One problem is assessing the completeness of a tool's exploration abilities. There is no clear bound on an analyst's ability to explore graph data and produce findings. But, if the atomic level functions to manipulate and represent data during analysis are enumerated,

then assessing which tool is "better" in an exploration task is possible because then the functions each tool offers can be compared.

Another problem is replication, an important tenet in scientific studies. Usually it is difficult to find precise documentation for how a visualization was created to display results of a study. This lack of an analytical instruction set makes reproduction of the visualization for other datasets difficult. Furthermore, in visualization tools, if a tool is centered around, for example, biological data, then there is no clear translation technique to replicate the tool's benefits and functions for another application, such as finance. A visual tool created for biological data may be useful for a financial analyst.

The underlying cause for these problems is the lack of a systematic method to design graph tools. There is no common language that defines the basic graph data manipulation actions in visual analytics. We present a visual analytic graph algebra to meet this void. The visual analytic graph algebra is part of a framework with the following components: (1) a formalized graph model, (2) an expressive predicate language, and (3) an algebra with associated operators.

The purpose of the operators is to manipulate the raw graph data before applying a visualization scheme. The selection operator zooms into a region of interest. The aggregation operator manipulates the data resolution with supernodes. The labeling operator tracks noteable information collected during an analysis, which is useful during lengthy analyses. And finally, the Visual Operator incorporates the interactive aspect of visual analytics in a work flow, which makes a visualization creation stage as flexible as the data manipulation stage.

The benefits of an algebra are now we can systematically replicate, compare, and assess graph visual analytics. Thus, a visual analytic algebra facilitates the production of graph information analysis.

Since the presentation of an algebra can be theoretical, to place it into practical applications we present an implementation of the algebra through Cytoscape [15], a graph visualization tool. This implementation is one example of how to implement the algebra. We will test the algebra's abilities to replicate and share graph analysis through a user study.

We demonstrate an example application of the algebra to analyze high throughput biological datasets. We reproduce statistical findings from previous genomic and metabolomic studies related to prostate cancer progression through visualizations created with the algebra. We also present optimization methods for repeated workflows to create a target visualization. We explore the visual analytic graph algebra rewrite rules which will result in fewer graph matching function calls.

The remainder of the chapter is structured as follows: We begin with related work specific to this chapter, Section 3.2. Next, we describe graph model, Section 3.3. Then we present the predicate language and operators, Sections 3.4 through 3.8. Next, we demonstrate an implementation of the algebra, Section 3.9 and present the user study, Section 3.10. In Section 3.11, we show an application of the algebra to analyze a high-throughput biological dataset. In Section 3.12 we discuss repeat workflow optimization techniques.

3.2 Related Work

In this section, we present related work on visual analytic challenges, optimization, and biological background. We present the biological background for the application study used in this chapter.

3.2.1 Visual Analytic Challenges and Theory

Recently there has been attention on the incomplete visual taxonomy [35] available to visual analysts. Also, there is a challenge in the visual analytic community to develop

frameworks and languages for visual analytic systems [29], [52], as well as building statistical analysis tools for visualization systems [62]. A number of approaches have been put forth to address the issue of coordinated multiple visualizations of data sets [50], [56]. Coordinated view systems are orthogonal to the visual analytic algebra. Our aim is to provide a formal visual analytic model to manipulate graph representations combined with interactive visualization techniques. Since the scope of visual analytics includes raw data manipulation, there needs to be a bridge between the data manipulation and visualization stages.

3.2.2 Optimization

Several techniques to speed graph matching have focused on designing clever indexes on a graph which can be used to filter nodes which are unlikely to match the query graph. For example, [65] uses a neighborhood indexing method based on vertex shortest path information. There has been some studies to compare different disk based graph indexing techniques by implementing different indexing methods on the same system in iGraph [27]. A variety of techniques are tested: gIndex, GF-Index, Tree+Delta, SwitchIndex, and FindFeatures. FindFeatures indexes structures based on frequent substructures found. Orthogonal to graph indexing is to perform approximate graph matching [53], which may be faster than exact querying. We can incorporate these methods within the Optimization Engine to benefit from faster graph querying.

Aside from graph indexing methods to speed the matching process, optimization of visual exploration requires understanding interaction costs in information visualization. In [34] a framework of interaction costs is proposed based on Norman's Seven Stages of Action to facilitate study and based on examining a collection of information visualization research papers, they suggest ways to narrow cost of execution and evaluation of a visualization tool. [57] proposes a general computation framework that derives a set of

data transformations to optimize the quality of a target visualization on general data sets. In [30] optimization techniques for visualizations are implemented through parallel programming methods. VisTrails [12] is a system which provides provenance management for exploratory tasks on visualization systems, with workflow rendered in a spreadsheet-style. Compared to these visual exploration optimization methods, the work presented in this chapter presents batch optimization methods on a saved workflow, to reproduce the target visualization faster for possibly updated parameters in the data transformation.

3.2.3 Biological Information Visualization

In the high-throughput biological dataset application we reproduce previous study findings through visualization. We briefly discuss select analytic techniques to study high-throughput biological datasets. With the wide variety of biological research, there is a wide variety of visualization tools to analyze these problems. For example, [40] presents a multiscale synteny browser to visualize and analyze relationships in syntenic data. In [43] an interactive graph display is presented which can replace and aid the manual work in the genome assembly process. In [33] a web-based visual interface is used to navigate through large microarray data sets. Heatmaps are a popular tool used to identify expression level changes among different groups. Furthermore, there are many products such as [21], [19] that are used to visually explore and find concepts related to a list of genes. Similar to these works, the visualizations we present in this paper can be used to explore data as well as present it.

3.3 Model

In this section we describe the attributed graph structure and its computed and intrinsic attributes. We also define the compositional functions, which are used to create computed attributes.

3.3.1 Structure

The central object of interest is an attributed graph, defined as $D = [G, X]$, where $G = (V, E)$ denotes a graph with vertex set $V \in \mathcal{V}$, edge set E, and attribute structure X. Each node $v \in V$ has a unique id attribute with a value assigned by an id label function λ. For clarity, we refer to each node by its id. For example, v_i is the node with $id = i$. Since the network can be multi-edge, the edge set can be described with $E \subseteq (V \times V, \mathbb{N})$. Each edge can be referenced uniquely by its incident nodes and the identification value (e.g. $e_{i,j,n} = (v_i, v_j, n)$). In the case of a simple network, we use $e_{i,j}$.

The attribute structure X associated with graph G has three components: $X = [X_V,\ X_E,\ X_G]$, where X_V contains node attributes (e.g. in- and out-degree), X_E edge attributes (e.g. edge betweenness, direction) and X_G graph attributes (e.g. diameter).

3.3.2 Attributes

Each attribute has a label, type, and a value. The label denotes the name of the attribute. For example: $color$, $last\ name$, and age are example labels with possible values $green$, $Smith$, and 42, respectively.

The attributes can be either an intrinsic or a computed type. Intrinsic attributes are independent features that stay the same even if other features or the graph topology changes. For each intrinsic node attribute i, we can express it based on the following mapping $X_V^i : V \to \mathcal{S}^{|V|}$, where the set $\mathcal{S}$ can take value in $\mathbb{R}, \mathbb{N}$ or any predefined value sets. The same applies to edge and graph intrinsic attributes.

Computed attributes are features that change as the graph's topology changes. We give an example network to illustrate how to depict the graph structure in our formal notation.

Example III.1. (Cell Phone Network) To describe a cell phone network, we can use a node to represent a cell phone and an edge represents a call between two phones. This is a multi-

edge directed network. The attributed graph for this data set is: $D_{phones} = \{G = (V, E)$, $X = (X_V = X_{phone\_id}, X_E = (X_{date}, X_{duration}, X_{tower}, X_{direction\_of\_call}), X_G = ())\}$.

The attributes in X are all intrinsic attributes describing features of their respective elements. The $phone\_id$ is the cell phone's unique identification number. The edges have several attributes including the date, duration, cell phone tower, and the direction in which the call was made. The direction attribute represents the directed nature of this graph. □

3.3.3 Composition Functions

To obtain computed attributes there exists a collection of composition functions $\mathcal{F}$. Since there are a variety of user-defined composition functions, we focus only on their core requirements. A composition function must combine attributes from an input set and generate a uniquely named computed attribute output set. If a function satisfies these requirements, we consider it in the class of composition functions.

For any $f \in \mathcal{F}$, we can express it as $f : \mathcal{Q}^{|*|} \times \cdots \times \mathcal{Q}^{|*|} \to \mathcal{P}^{|*|}$ with $* \in \{V, E, G\}$ where $\mathcal{Q}$ can take a value in $\mathbb{R}, \mathbb{N}$ or intrinsic attribute space $\mathcal{S}$ and $\mathcal{P}$ can take a value in $\mathbb{R}, \mathbb{N}$, or any predefined value sets.

Without loss of generality, we list some composition functions.

- 0 or 1 indicator function is a composition function with $\mathcal{Q} = \mathcal{S}$ and $\mathcal{P} \in \{0, 1\}$.
- Numerical aggregation functions, such as sum, average, and min/max, over a collection of attributes are composition functions.
- A function of composition functions is also a composition function; therefore, the class of composition functions is closed.

The following example shows how to construct a composition function to produce degree, which is a node computed attribute.

Example III.2. (Counting Function) Given a simple edge network, a possible set of composition functions to compute the degree of a node, v_i, given its neighbor list $\mathcal{L}_{v_i}$ is COUNT$(\mathcal{L}_{v_i}) = \sum_{v_j \in V} 1_{\{v_j \in \mathcal{L}_{v_i}\}}$ summation over a list of indicator functions.
The neighbor list can be constructed based on graph information $G = (V, E)$ where $\mathcal{L}_{v_i} = \cup_j \{v_j | e_{i,j} \in E\} \bigcup \cup_k \{v_k | e_{k,i} \in E\}$. $\square$

3.4 Predicate Language

In a graph analysis, the analyst starts with an intent, which is represented as a predicate. The predicate allows the analyst to describe structural and attribution conditions on a graph structure she would like to find and study (possibly resulting in manipulating the data). We use a graph matching function, γ, to search the graph for the predicates. The objects we find are called $witnesses$. They represent the instances of the predicates occuring in the graph. In this section, we go into detail about these structures and the graph matching function. They will be used heavily in the algebra operators.

3.4.1 Predicate

A necessary requirement in any analytical graph tool is the ability for an analyst to express the graph structure she would like to further examine or manipulate. A predicate is such structure; specifically, it is a graph structure with conditions on attributes.

Formally, a predicate tuple is $p = (V, E, \mathbb{X}_V, \mathbb{X}_E, \mathbb{X}_G, !E)$. The $\mathbb{X}_V, \mathbb{X}_E$, and $\mathbb{X}_G$ terms in the predicate are condition lists on the attribute values in X_V, X_E, X_G, respectively. The form for a condition in $\mathbb{X}_V$ is $X_{v.a.i}\ op\ value$ where $X_{v.a.i}$ is the attribute a for node $i \in V$, and op is any relational operator. Similarly the form for a condition in $\mathbb{X}_E$ and $\mathbb{X}_G$ is written as $X_{e.b,j}\ op\ value$ and $X_{g.c}\ op\ value$, respectively, where $j \in E$, b is an edge attribute, and c is a graph attribute.

The $value$ is a constant or a reference to any attribute's value, irrespective of the at-

tribute type (node, edge, or graph). Wildcards are allowed in $value$.

The form for a node attribute reference for attribute x on node $k \in V$ is $X_{v.x.k}$. Note that k is any node in V, $k \neq i$, the node on which we place the condition. An edge attribute reference for attribute y on edge $j \in E$ is written as $X_{e.y.j}$. Finally, a graph attribute reference for attribute z is written as $X_{g.z}$. The following is an example predicate showing attribute conditions referencing other nodes.

Example III.3. (Cross-Referencing) To express the predicate shown in Figure 3.1 where a node with $Phone\_id = 200$ and $degree = 40$ has two neighbors. The neighbors' degree is described such that one has a higher degree than the other. The predicate tuple is $p = (V = (v1, v2, v3), E = (e_{1,2}, e_{1,3}), \mathbb{X}_V = ((X_{v.degree.1} = 40), (X_{v.Phone\_id.1} = 200), (X_{v.degree.3} > X_{v.degree.2})), \mathbb{X}_E, \mathbb{X}_G, !E)$. □

In the tuple $!E$ is the excluded edge list. An $exclusion$ on edge e specifies that e must not exist in the graph G. Furthermore, if G is defined on a "universe" G', then the $excluded$ edge must also not exist in G'. What this universe should be is usually clear in context, typically an initial input graph before any selections have been applied. Where the universe is not clear, we will choose it to be the graph G itself. Note that $!E$ is subset of the complement of E.

Definition III.4. (Exclusion) Given predicate $\rho = (V, E, \mathbb{X}_V, \mathbb{X}_E, \mathbb{X}_G, !E)$, and a graph $G = (V_G, E_G)$. If $e_{i,j} \in !E$ then $e_{i,j} \notin E_G$. Furthermore, in a closed universe U, $\forall S$ where $G \subseteq S = (V_S, E_S)$ then $e_{i,j} \notin E_S$. □

Multiple conditions on an attribute express ranges as the following example shows.

Example III.5. (Ranges) To express the range that node attribute a must be between values x and y, we use the following conditions: $\{(X_{v.a.i} \leq x), (X_{v.a.i} \geq y)\} \in \mathbb{X}_V$. □

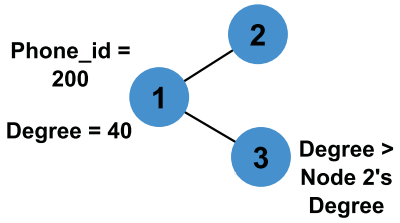

Figure 3.1: The figure shows a simple predicate corresponding to the cross-referencing example. Conditions are placed upon the attributes on the nodes. It is possible for one condition to reference the attribute on another node.

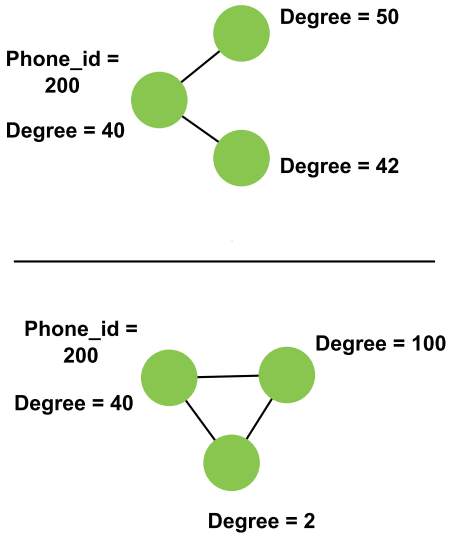

Figure 3.2: These graph structures satisfy the predicate shown in Figure 3.1. They are two possible instantiations of the predicate.

Figure 3.3: The two predicates show similar graph structures. However, the figure on the left has an excluded edge between nodes 2 and 3. Figure 3.4 shows the result of this excluded edge on possible witnesses given an input graph.

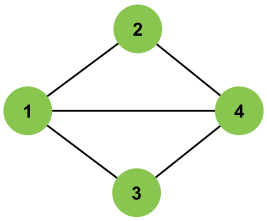

Figure 3.4: The figure above is an input graph. Given the two predicates shown in Figure 3.3 we describe the different witnesses existing in this input graph. The sets of nodes which induce a witness to the excluded edge predicate are: 1,2,3 and 2,4,3. The predicate without the excluded edge results has more witnesses: 1,2,3; 1,2,4; 1,4,3; 2,4,3; and more. Since there is an edge between node 1 and 4, this the witness 1,2,4 does not satisfy the predicate with the excluded edge.

3.4.2 Witness

In the previous section we described how to define a predicate. The actual realization of a predicate in an input graph is called a $witness$. In this section, we will give a formal definition of a witness and several example witnesses given a predicate. It is careful to see that the translation of an analyst's intent, which we represent as a predicate, can result in several witnesses, which may not be identical to one another.

Definition III.6. (Witness) Given an attributed graph $D = [G = (V, E), X = (X_V, X_E, X_G)]$ and a predicate $p = (N, M, \mathbb{X}_N, \mathbb{X}_M, \mathbb{X}_G, !M)$, D is a witness to p if there is a bijection mapping function, $f_{D,p}$, between the vertices in V and N such that,

- $e_{i,j,k} = ((v_i, v_j), k) \in E$ if and only if $m_{i,j,l} = ((n_i, n_j), l) \in M$ where $v_i = f_{D,p}(n_i)$ and $v_j = f_{D,p}(n_j)$ and $k = f_{D,p}(l)$
- $\forall (X_{n.i.x}\ op\ value) \in \mathbb{X}_N$, $(X_{v.f_{D,p}(i).x}\ op\ value)$ holds in D, where $i \in N$ and x is a node attribute label.

and an injection edge attribute assignment function, $h_{D,p}$, between the edges E and M such that $\forall m_{i,j,k} = ((n_i, n_j), k) \in M$

- $\forall (X_{m.j.y}\ op\ value) \in \mathbb{X}_M$, $(X_{e.h_{D,p}(j).y}\ op\ value)$ holds in D, where $j \in M$ and y is

an edge attribute label.

the conditions on $\mathbb{X}_G$, must be satisfied in X_G,

- $\forall(X_{g.z}\ op\ value) \in \mathbb{X}_G$, $(X_{g.z}\ op\ value)$ holds in D, where z is a graph attribute label.

and finally, if in a closed universe U, $\exists\ D' = [G' = (V', E'), X' = (X_{V'}, X_{E'}, X_{G'})]$ such that $D \subseteq D'$

- if $m_{i,j,k} = ((n_i, n_j), k) \in !E$ then $e'_{i,j,w} = ((v'_i, v'_j), w) \notin E'$ where $v'_i = f_{D',p}(n_i)$ and $v'_j = f_{D',p}(n_j)$. $\square$

Both structural and attribute properties of D, and the predicate, p, must match for D to be a witness of p. There must be a bijective mapping between the vertices and edges in the predicate structure and the witness. In other words, there must be an isomorphic matching between the node and edge identifications.

We show some examples of predicates, witnesses, and $excluded$ edges.

Example III.7. (Witnesses) Figure 3.2 shows two example witnesses for the predicate in Figure 3.1. Both satisfy the structural and attribute conditions although these witnesses clearly are not identical graphs.

Example III.8. (Predicate with Excluded Edges) Figure 3.3 shows two predicates. The predicate on the left has an $excluded$ edge between nodes 2 and 3. This predicate is written as $p_1 = (V = (v1, v2, v3), E = (e_{1,2}, e_{1,3}), \mathbb{X}_V, \mathbb{X}_E, \mathbb{X}_G, !E = (e_{2,3})$. Note that an edge is denoted with two indexes when the third one is not relevant. The predicate on the right is a similar predicate but does not have an $exclusion$ between nodes 2 and 3. This predicate is written as $p_2 = (V = (v1, v2, v3), E = (e_{1,2}, e_{1,3}), \mathbb{X}_V, \mathbb{X}_E, \mathbb{X}_G, !E = ())$. $\square$

Example III.9. (Excluded Edges) The difference between predicates shown in Figure 3.3 can be seen by the supergraph in Figure 3.4. The nodes in the input graph that induce

witnesses to the predicate with an excluded edge are: $\{1, 2, 3\}$ and $\{2, 4, 3\}$. There are many more sets of nodes that induce witnesses to the predicate without the excluded edge. For example, a few are: $\{1, 2, 3\}$; $\{1, 2, 4\}$; $\{1, 4, 3\}$; and $\{2, 4, 3\}$. Due to the edge between nodes 1 and 4, the subgraph induced by the set of nodes $\{1, 2, 4\}$ cannot be a witness to the predicate with the excluded edge. □

3.4.3 Graph Matching Function

The process of pairing a predicate to a witness is called graph matching. Graph matching is a well-studied problem in graph analysis. In terms of Visual Analytics, it is useful to see how a predicate matches a witness in an input graph. For example, a predicate might have many witnesses in an input graph. How do these witnesses compare to each other? Where are these witnesses in relation to each other in the input graph? By knowing the details of how a witness matches a predicate, an analyst can answer similar questions.

In this section we describe a graph matching function, γ. The specific graph isomorphism algorithm can be replaced with any state-of-the-art algorithm. We focus on the input and the outputs of the graph matching function.

Definition III.10. Given an attributed graph D and a predicate p, graph matching γ outputs

- A list of witnesses found in D. In the case of duplicate witnesses, a single arbitrary witness is returned from the duplicate set.
- A model witness X which is an attributed graph instantiation of the predicate's structure and attribute conditions.
- A set of mapping lists for each witness to X.

Often, the same set of nodes in the graph can match at multiple positions in a structural predicate. This is trivially true when the condition applied at each node is the same. In

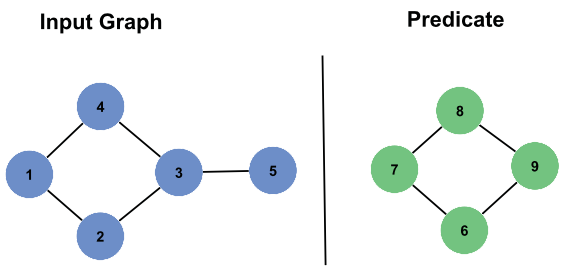

Figure 3.5: The Graph Matching function takes two inputs: a graph and a predicate. Given these inputs, the graph matching function will find witnesses that satisfy the predicate within the input graph.

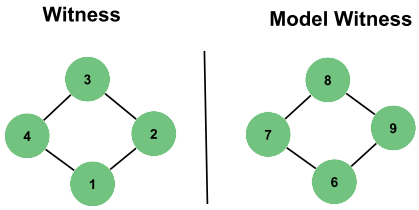

Figure 3.6: Given the inputs to the graph matching function shown in Figure 3.5, the function will return three types of output. First there exists one witness in the input graph. Second, a model witness is returned that maintains the predicate structure. In this case, the model witness is identical to the witness. But it is possible the witness contains an edge between nodes 1 and 3 and still be a witness to the predicate. The last structure returned is a mapping list for the witness to the model witness. The mapping list is useful for the analyst to see how the witness matches the predicate. The mapping list in this case is: $\{1 \to 6, 2 \to 7, 3 \to 8, 4 \to 9\}$

this case, every permutation of node matches is a "new" way to satisfy the given predicate. This can result in an unacceptably long list of matches. To avoid this eventuality, we consider two witnesses to be duplicate if they comprise the same set of nodes and edges, even if the matching to the structural pattern is different. The mapping lists save how each witness satisfies the predicate.

Example III.11. (Gamma function) Given the two inputs to the graph matching function shown in Figure 3.5, the matching function returns a single witness, a model witness, and a mapping list of the witness to the model witness. The outputs are shown in Figure 3.6.

3.5 Selection Operators

There are two selection methods that operate on sets of graphs: set and element selection. Set selection reduces the number of graphs in a set. Element selection reduces the nodes and edges in each graph.

3.5.1 Set Selection

Set selection filters a set of attributed graphs that do not have at least one witness for the given predicate. For example, an analyst may start with a large collection of graphs but she is only interested in the graphs with a diameter larger than 20. Set selection is applied to focus on the interested set.

Definition III.12. (Set Selection) Given a collection of graphs and their attributes, $\mathcal{D}$, a set selection with predicate α, is $\sigma_{set,\alpha}^{extract}(\mathcal{D}) = \{D \in \mathcal{D} | \text{there exists a witness in } D \text{ for } \alpha\}$. As a subroutine, the graph matching γ function is called to determine if any witness exists for each of the attributed graphs. □

Example III.13. Figure 3.7 shows a set of two attributed graphs. If we perform a set selection with a predicate that the average degree of the input graph be equal to 2, the result will be the first attributed graph. Each node in the result has a degree of 2. Since it is the only input graph that satisfies the predicate, it is the only graph returned.

3.5.2 Element Selection

Similar to set selection, element selection zooms into points of interest. In this case, nodes and edges within graphs are selected. For example, an analyst may have several graphs but would like to only focus on the nodes with a degree greater than 5, in other words, remove all the other nodes. She would use element selection.

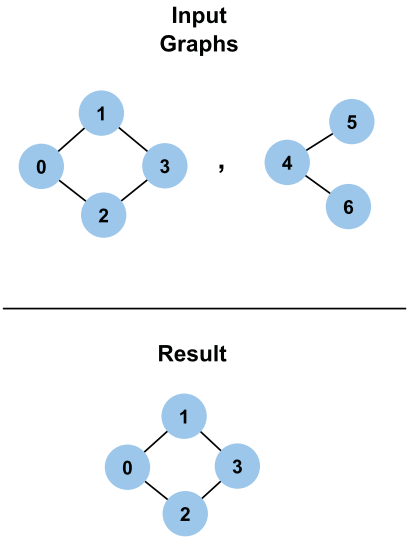

Figure 3.7: Two input attributed graphs are displayed. If the predicate to the set selection function is that the graph attribute, average degree, be equal to 2, then the result from set selection is the input graph shown on the left. The input graph shown on the right has an average degree of 1.67.

Given a set of attributed graphs and a predicate, element selection creates a new attributed graph for each witness match found. Since a new attributed graph is created, the previous computed attributes may not apply to the new topology of the graph. Therefore, computed attributes are updated for the new attributed graphs.

Definition III.14. (Element Selection) Given a collection of attributed graphs, $\mathcal{D}$, an element selection with predicate α, is $\sigma^{selection}_{element,\alpha}(\mathcal{D}) = \cup_i \{W'_i\}$ that satisfies

- $D_i = [G_i, X_i] \in \mathcal{D}$
- $W_i = \{W_{i.\alpha.1}, \cdots, W_{i.\alpha.k}\}$ for k = number of witness of α on D_i

where each $W_{i.\alpha.j} = [G_{i.\alpha.j}, X_{i.\alpha.j}]$ is the j^{th} witness of α on D_i.

The attributes in each witness, $W_{i.\alpha.j}$, are as follows:

1. All graph attributes from D_i carry through to $X_{i.\alpha.j}$
2. All node and edge attributes in D_i which are attributes of the nodes and edges in $W_{i.\alpha.j}$ carry through to $X_{i.\alpha.j}$

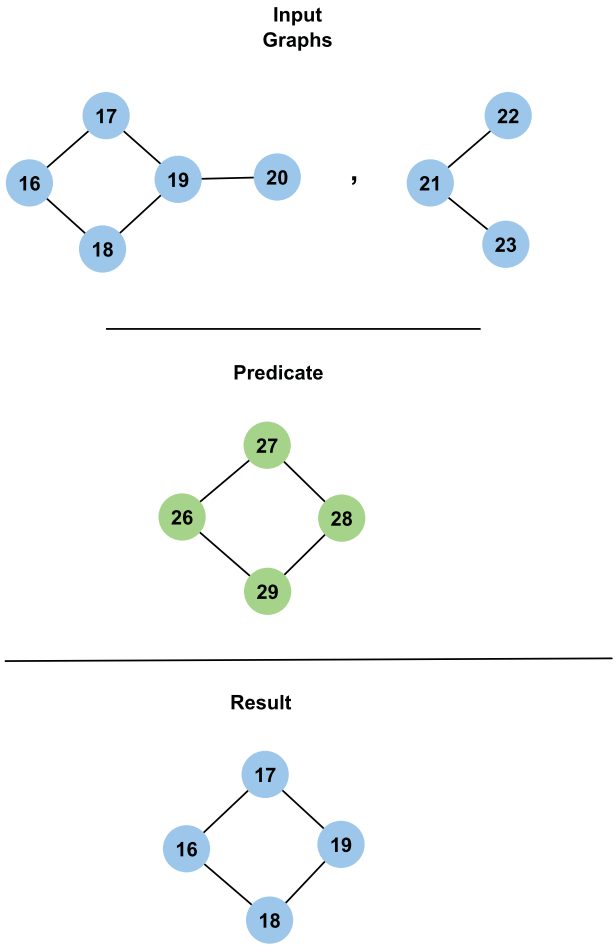

Figure 3.8: Given the two input graphs and the predicate shown above, the resulting attributed graph after an element selection call is shown. The input graph contains a witness for the predicate, namely the witness induced by the node set (16,17,18,19). The input graph on the right has no witness for the given predicate. The graph matching function is called to determine if a witness exists in an input graph.

3. All computed attributes in $X_{i.\alpha.j}$ are updated □

Example III.15. Figure 3.8 shows two input graphs and a predicate. Element selection returns all structures that match the given predicate structure. The only attributed graph returned is from the input graph on the left. No witness for the predicate exists in the input graph shown on the right.

3.6 Aggregation Operators

We use aggregation to reduce the complexity of a graph by combining structures with similar patterns. These patterns are stated as a list of predicates.

Aggregation operates on both the set level and the element level. At the set level, aggregation unions the attributed graphs together that satisfy a predicate. The result is a new set of attributed graphs, one for each predicate. At the element level, aggregation merges the nodes or attributes together that satisfy a predicate in the list.

3.6.1 Set Aggregation

Set aggregation performs a union over the elements of the attributed graphs that satisfy a given predicate. For each of the predicates, set aggregation checks each of the n input graphs for at least one witness.

For the input graphs that satisfy the predicate, set aggregation will union their elements (graph and attributes) and produce a single output for the predicate. The aggregation performs as a union of a set of disjoint attributed graphs. The elements of the inputs are given a new unique identification in the new outputs.

Definition III.16. (Set Aggregation) Given a set of predicates, $\beta = \{\beta_1, \cdots, \beta_m\}$, and a set of attributed graphs, $\mathcal{D}$, set aggregation is defined as $\phi_{set.\beta}(\mathcal{D}) = \mathcal{D}' = \{D'_1, \cdots, D'_m\}$. where each D'_i are the union input attributed graphs that satisfy predicate β_i.

Please refer to the appendix for the set aggregation algorithm.

Example III.17. Sometimes to analyze a phone call network over a long period of time, analyze merge multiple days to compare groups of days. In this example, assume we have a set of phone call networks, each representing a single day (with a graph attribute $X_{G,day} = i$), over a four day period, we wish to aggregate days one to two and aggregate days three to four. In this case we call $\phi_{set.\beta}(\{D_{day1}, D_{day2}, D_{day3}, D_{day4}\})$ where β is the predicate set with one predicate on days 1-2 and the other on days 3-4. This set aggregation produces the following set $\{D_{days1-2}, D_{days3-4}\}$ $\square$

3.6.2 Element Aggregation

Element aggregation merges the nodes and edges of an attributed graph by groups. The groups are described by a set of predicates. There are two types of groups: group by structure or group by all structures.

In group by structure, element aggregation creates a new node with a unique id for each witness found for each predicate. The new node represents a merged supernode. The attributes of the new node are aggregated values of the nodes contained in the witness. The supernode also maintains the connections to the model witness node and to the nodes in the graph which the aggregated nodes connected to before being combined into the supernode.

Group by all structure is similar to group by structure except supernode is not created for each witness found. Instead, one supernode is created to represent all the witnesses for a predicate.

Definition III.18. (Element Aggregation) For a given predicate list β= $\{\beta_1, \cdots, \beta_m\}$, an attributed graph set $\mathcal{D}$, element aggregation is defined as $\phi_{element.\beta.type}(\mathcal{D})$ where $type$ is either by structure or by all structures. Element aggregation modifies the input graphs into

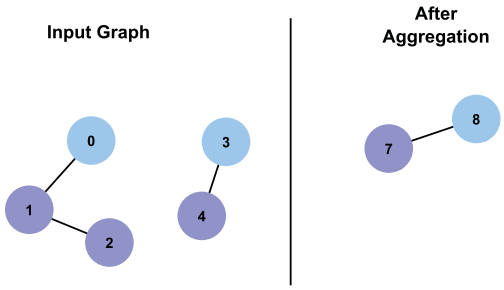

Figure 3.9: An element aggregation by all structures is performed on the input graph. The result is the blue nodes are merged into one group, and the purple nodes into a second group.

aggregated graphs as described in the Appendix.

We show the difference between aggregation by structure and by all structures with examples.

Example III.19. (Element aggregation by all structures) Figure 3.9 shows the effect of an element aggregation on an input graph. The predicates we pass are $\alpha = \{\alpha_1 = \{V = \{V_1\}, X_{v.color.1} = blue\}, \alpha_2 = \{V = \{V_1\}, X_{v.color.1} = purple\}\}$.

After the aggregation, nodes 1, 2, and 4 are merged together, which is α_1's witness group; and, nodes 0 and 3 are merged together, which is α_2's witness group. The computed attributes for the new merged nodes 7 and 8 are aggregated values from their merged nodes. The attribute type determines how to aggregate the values. For example, numbers may be averaged together.

Model witness nodes 9 and 5 are created to save the predicate match structure and attribute conditions. Their purpose is to remind the analyst in the future why these nodes were aggregated. There are directed edges from the supernodes to their respective model witnesses. This added attribute and structural information differentiates a supernode from non-aggregated nodes. □

Example III.20. (Element aggregation by structure) In Figure 3.10 there is an input graph

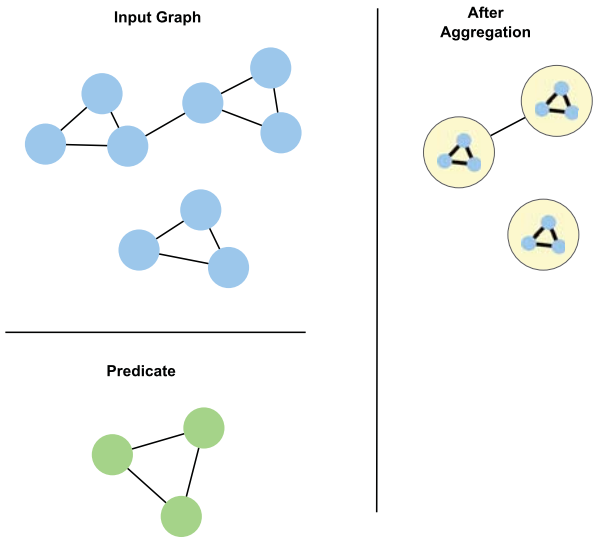

Figure 3.10: After an element aggregation by structure, the attributed graph becomes only three nodes.

containing three triangular structures. An element aggregation *by* structure given the predicate shown will create three merged nodes. The two merged nodes that are connected to each other result from the connection between two of the triangular structures in the input graph. In the "after aggregation" picture, we can see the merged triangular structures within the new nodes. Figure 3.11 shows how the merged structures within the new nodes are stored. This is done by creating a model witness for the predicate. The new nodes point to the model witness structure.

3.7 Labeling

Just observing a network can lead to many mental notes. But for every observation, it is too cumbersome to open a new branch of analysis. In terms of our algebra, we may prefer not to have to perform a selection and aggregation to create a new graph. Sometimes, simply making a note on the current graph will suffice. Then as analysis continues, these notes could be saved and easily tracked. The structure of this "mental note" on a graph leads us to our next operator: Labeling.

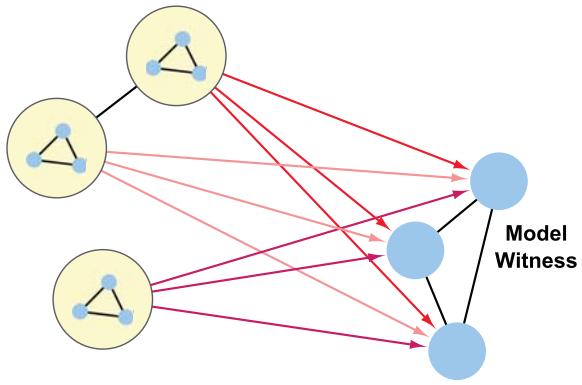

Figure 3.11: The three aggregated nodes in the merged graph of Figure 3.10 point to a model witness. The purpose of this model witness is the retain the predicate structure, the reason for the aggregation, to understand the analytical process.

Labeling does not create a new graph, but it creates structured attributes based on a process similar to the element selection. The inputs to the labeling operator are a set of attributed graphs and a predicate.

As a result of labeling, the attributed graphs that contain a witness are modified to include edges to a model witness (for the predicate). The model witness, similar to element aggregation, is given a new group id. This group id is used to identify it as an informational structure within the graph (not to be included in selections).

Below is a formal definition of labeling:

Definition III.21. (Labeling Operator) Given a collection of n graphs and their attributes $\mathcal{D}$, and predicate $\alpha = (V_\alpha, E_\alpha, X_{V_\alpha}, X_{E_\alpha}, X_{G_\alpha})$, a labeling, $\sigma^{label}_{element,\alpha}(\mathcal{D})$ is: For each attributed graph D_i in $\mathcal{D}$ where there exists a witness for the predicate α,

1. Create the model witness structure X within D_i
2. Label X's nodes and edges with a unique group id (can be user defined)
3. For each witness W_j found in D_i use the mapping lists to create directed edges from the nodes in W_j to the nodes X. For the new edges created in W_j to X create two

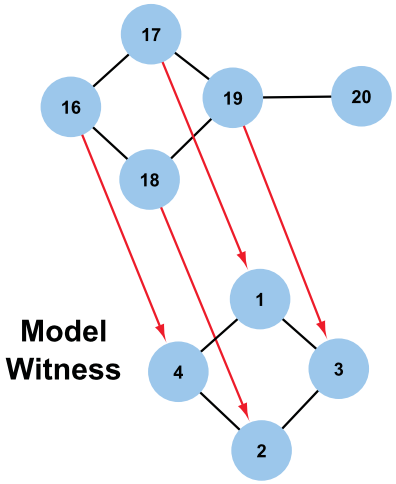

Figure 3.12: A labeling operator performed on the same input graph and predicate from the element selection example, Figure 3.8

attributes: one to denote that it is part of one witness structure and another for the unique group id of this model witness.

□

Example III.22. In Figure 3.8, we show how to create new graphs based on an input set and predicate after applying the element selection operator. Sometimes an analyst does not want to continually create more networks, rather within an exist network make notes and follow certain nodes of interest.

In Figure 3.12 we show how the labeling operator achieves this goal. Given the same input graph set and predicate from Figure 3.8, the result of the labeling operator is a model witness included into the graph (for the given predicate) and directed edges between the matched components of the input graphs. The model witness is not included in graph matching steps. □

3.8 Visualization Operator

The operators that we presented so far cover data manipulation and management for graph analytics. We began by describing an analyst's intent through a predicate structure. Then, we laid out the operators which select, aggregate, and label graph data. To relay the outcome of the operators to the analyst, there needs to be a visual component to the algebra. Intermittently between the data manipulation and management operators, the analyst uses the Visualization Operator, $\mathcal{V}$, which lets her visually interact with the algebra and share her work by producing visualizations. For example, to see the result of a set aggregation and make decisions on an overall trend, the graph needs to be visualized after the operator is applied. The main purpose of the Visualization Operator is to produce visualizations.

In this section, we define the Visualization Operator and give an application demonstrating its use. Unlike the previous operators, we leave the exact mechanics of the operation flexible. The ways to visually express information is endless. For the algebra to describe and compare multiple tools, the Visualization Operator cannot be limited to a set of visual encoding and rendering methods. Yet the algebra still needs to capture how a visualization is created.

To make this compromise, we define the Visualization Operator to take two inputs: a predicate, and a visual function. The output is a static visualization. The predicate describes the graph that is to be displayed. The visual function calls the tool specific function to create the visualization. But by providing it as an input, we save the mechanism for documentation. Possible visualization functions a tool may provide include:

- Different layout options (force directed, circular, etc...) to display a graph
- Coloring nodes by a rainbow gradient based on their degree value

These are functions that affect the display. The Visual Operator encapsulates the com-

bination of computation and visual rendering and encoding which produce a visualization. We will demonstrate the Visual Operator in practice with an application below.

3.8.1 Cytoscape Visual Display

In the previous subsection, we listed ”Coloring nodes by a rainbow gradient based on their degree values” as a function a visualization tool may provide. In this example, we use Cytoscape [15], which is an open source graph visualization tool. Cytoscape has a function to color nodes based on their degree values. We illustrate how the Visual Operator works in the instance this function is invoked.

Assume we loaded a phone call network data set into Cytoscape, shown in Figure 13. We would to see the distribution of degree, in this case, degree reflects the quantity of phone calls made to other phones. An analyst can use Cytoscape's coloring function to see which degrees occur most often in the graph. The visualization we want to produce is a graph with its nodes colored based on degree values.

We trace Cytoscape's color function through the Visual Operator (effectively assuming the analyst already selected other display features, such as the graph layout).

Action: The analyst selects the color function on the phone call network data, denoted as D1. Algebra:

1. The visual operator is called with a predicate describing D1, and “Color Function” as the mechanism: V(D1, “Color Function”). The following are the substeps V(D1,“Color Function”) encapsulates:
   (a) Node attribute Xdegree (storing a node's degree) is created through composition functions: See Example 2 for the algebra.
   (b) Node attribute Xcolor (storing a node's color) is created through a mapping composition function based on Xdegree.

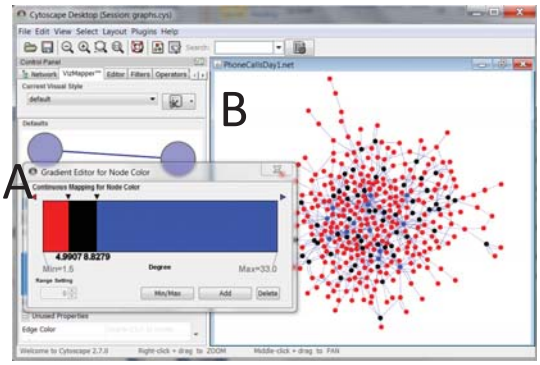

Figure 3.13: A continuous color gradient mapping to node degree is set on the phone data set from the VAST 2008 Challenge.

(c) Tool Specific visual encoding and rendering step. Since we cannot bound the software and hardware possibilities for different tools, we note this as Tool Specific in the algebra.

(d) Output is produced, as shown in Figure 13.

Tying a visual function in a tool with the Visualization Operator helps maintain the display information (Xdegree and Xcolor in this case). It might appear convoluted to turn the task of coloring nodes by degree into a set of algebraic expressions. Actually, it can become worse if there is no algebra to trask visual tasks. If it takes several visualization and data manipulation steps to reach an analytical decision, it will be difficult to replicate or verify the decision had it not been documented.

Also, without an algebra, comparing tools and methods is difficult. Additionally, the visualization becomes flexible. We broke the visualization task into manageable steps. We can change a substep (for example, change the composition function to compute degree), or simply rerun the visual task on a different input, without having to manually trace our work, because now we have it explicitly documented.

3.9 Implementation Example in Cytoscape

The presentation of the algebra so far has been through conceptual examples and theoretical definitions. In the preceding Visual Operator section, we started to show how we can connect the algebra formalisms to tool. In this section, we complete this connection by presenting a plugin, called the "Visual Analytic Algebra" plugin, for Cytoscape which actualizes the algebraic framework and operators.

3.9.1 Graph Model

Graph visualization tools, by design, support a graph model. Our graph model is composed of a collection of graphs, computed and intrinsic attributes, and composition functions. In this section, we walk through how each graph model component is realized in Cytoscape. Figure 3.15 shows a Cytoscape display window after several phone call networks have been uploaded. In this display, the left panel is called the "Control Panel," which contains several tabs. The plugin has added several new tabs to the "Control Panel." We summarize the "Control Panel" tabs, which relate to the graph model:

- The Network Tab displays a list of the network names. Through this tab the collection of phone call network graphs can be accessed.
- The Modify Attribute Types Tab, shown in Figure 3.14, displays the attribute labels for graphs, edges, and nodes. In this panel, the type of the attribute can be switched between intrinsic and computed.

3.9.2 Predicate Language

The predicate language includes a predicate structure, witness representations of the predicate, and a graph matching function. The predicate structure is a graph structure with conditions placed on its attributes. The witness represents occurrences of the predicate in

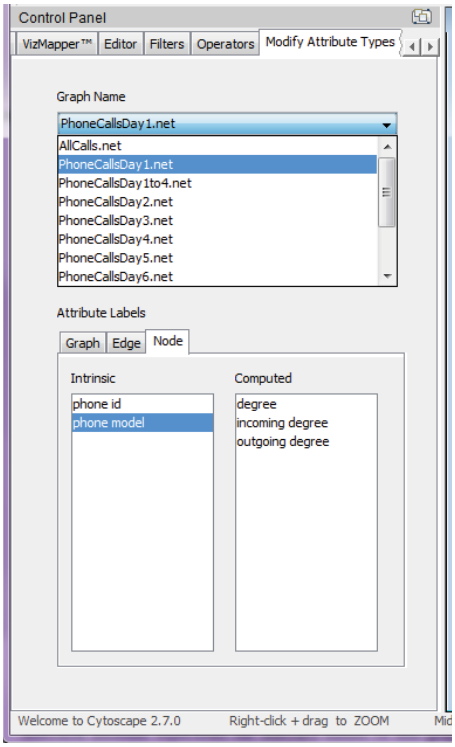

Figure 3.14: As part of our Visual Analytic Algebra, we define attributes as either computed or intrinsic. To allow the user to modify the type of an attribute we provide the following tab under the Control Panel. The user can view the attribute labels for nodes, edges, and the graph. To change the type from intrinsic to computed, the user can drag the attribute label from one type to another.

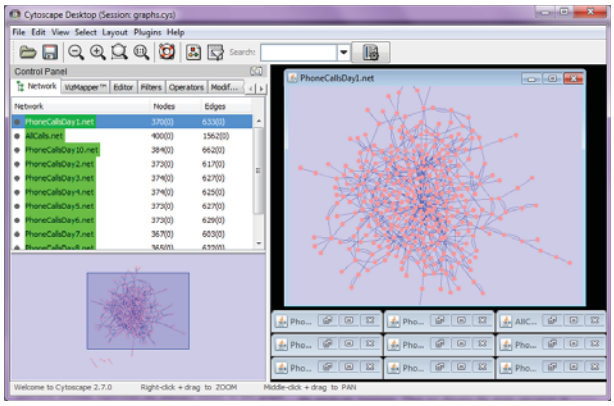

Figure 3.15: The main Cytoscape window is shown. There are three main components to its design. The network panel displays the network. The data panel displays node and edge specific attribute information. The control panel has several tabs to perform different functions on the network. One of these tabs is the network tab, that shows all the networks opened during a current Cytoscape analysis session. It allows the user to switch between different networks by saving them under different names.

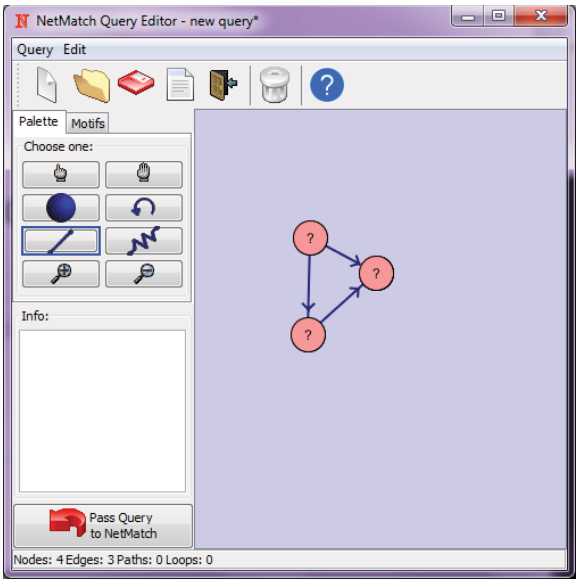

Figure 3.16: The plugin NetMatch allows a user to draw a query. In essence, a query is a predicate. Attribute conditions are possible on nodes.

a graph. To find witnesses given a predicate, we use the graph matching function. We implement these concepts through the plugin. The "Visual Analytic Algebra" plugin utilizes a plugin "NetMatch," which supports the predicate language. We list these features:

- A predicate structure is created through the "NetMatch Query Editor," shown in Figure Figure 3.16. An analyst can draw a new predicate and place conditional values on its attributes.

- To produce witnesses for a given predicate, on the "NetMatch Query Editor" window, we use the the "Pass Query to NetMatch" button, shown in Figure Figure 3.16. This procedure will result in the witnesses, shown in Figure 3.17. Each witness is displayed on a row and the columns indicate the number of its matches and the matching nodes (which corresponds to the mapping list of the graph matching function).

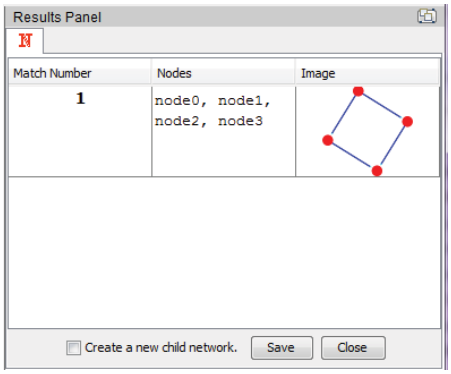

Figure 3.17: Using the NetMatch plugin, we show the result of matching the input graph and the predicate shown in Figure 3.5. The image shown is a witness found in the input graph. The Node column shows the mapping between the witness and the predicate. This figure is an example implementation of the Predicate, Witness, and Graph Matching features in the Visual Analytic Algebra.

3.9.3 Operators

We presented several operators, for example, selection, aggregation, labeling, and the visual operator. In the Visual Operator section, we gave an example of how to encapsulate the details of a visual function into the operator. In this section, we will show implementations of the selection and aggregation operators in the Cytoscape plugin.

We implement a direct way to perform set and element selection. The "Control Panel" has an "Operators Tab," shown in Figure Figure 3.18 . This tab includes options to select a set of networks, the predicate (which is designed in the NetMatch Query Editor), and the specific operator to perform. The following examples demonstrate the different operators, shown in Figure 21.

Example III.23. (Set Selection) When there are several graph displays open in a Cytoscape session, a useful way to reduce it to a few of interest is through set selection. We apply a set selection operator on the highlighted graphs in the "Operators Tab." The results are shown in Figure 3.19. The networks that satisfy the predicate are maximized and shown in the display panel. The non-matching networks are minimized into the back-

ground of the display panel.

Example III.24. (Element Selection) An element selection will filter the nodes of interest within a graph. We utilize the“Operators Tab” again. Figure 3.20 shows (1) the input graph, (2) the predicate, and (3) the results from element selection. Similar to set selection, the analyst needs to first specify the input graph and predicate. Element selection requires the graph matching function to prepare results. In this example, three witnesses are found. The analyst is given the option to create new networks for the witnesses found by selecting a witness and the “Create a new child network” option. All match information is possible to save for documentation.

Example III.25. (Aggregation) An element aggregation creates new merged nodes within the graph. Figures 22 and 23 show element aggregations by all and per structures, respectively. In these figures, we show the Cytoscape realization of the element aggregation example of Figure 3.10.

Since Figure 3.22 is an aggregation by all structures and all nodes in the input graph are witnesses to the predicate, they become merged into a single supernode. In Cytoscape the model witness is displayed within the supernode. This feature helps distinguish supernodes from lower granularity nodes. Figure 3.23 has the same aggregation by structure result as shown in Figure 3.10. In Cytoscape, the aggregated graphs are created as “child” graphs to the input graph. In the network tab, the child graphs will be listed below their parent graph.

In [46], we show more examples on using the algebra within Cytoscape to solve the VAST 2008 Challenge.

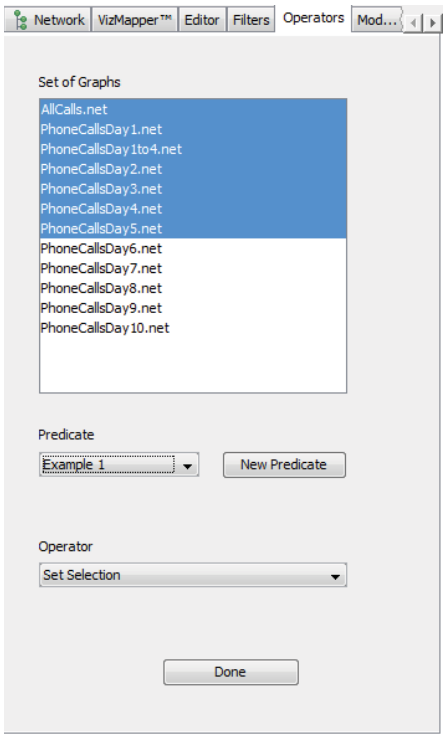

Figure 3.18: The Visual Analytic Graph Algebra Plugin also includes an Operators Tab in the Control Panel. Here the analyst can select a set of networks, the predicate list, and the operator to apply to the graphs. In this figure, the set selection Operator has been selected with the Predicate from Figure 3.1.

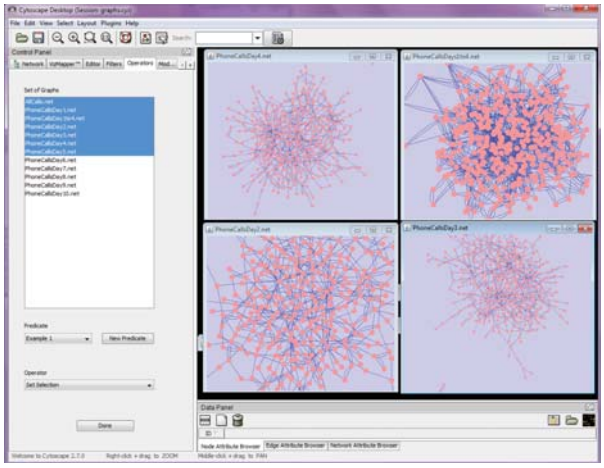

Figure 3.19: This figure shows an example implementation of set selection in Cytoscape. Once the ”Done” button has been clicked, the result of the set selection is the set of graphs that satisfy the predicate are opened and displayed in the Display Panel. In this figure, four graphs are displayed.

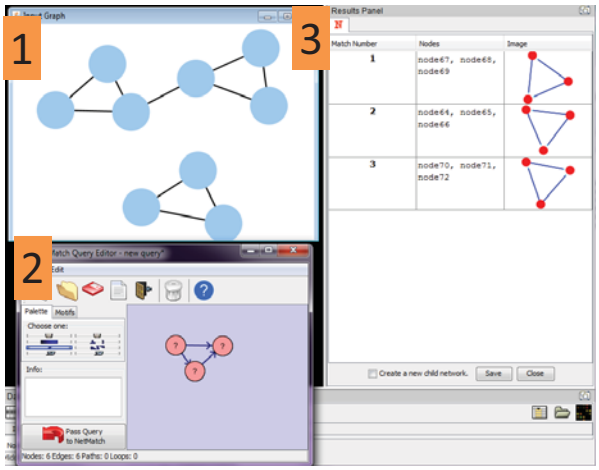

Figure 3.20: This figure shows (1) the input graph, (2) the predicate, and (3) the results from element selection.

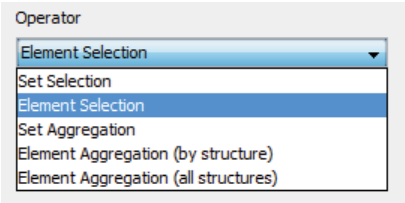

Figure 3.21: On the Operators Tab, the drop down menu shown has all of the operators available for easy access for the analyst.

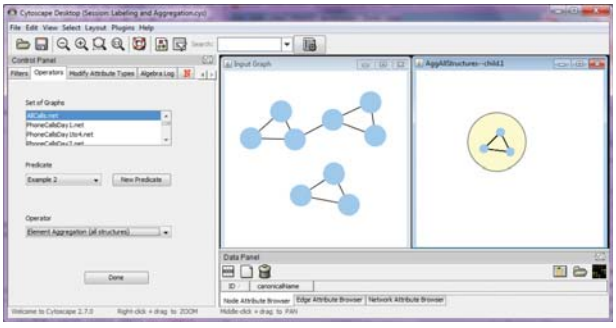

Figure 3.22: A Cytoscape Aggregation by All Structures is performed on the input graph shown on the left.

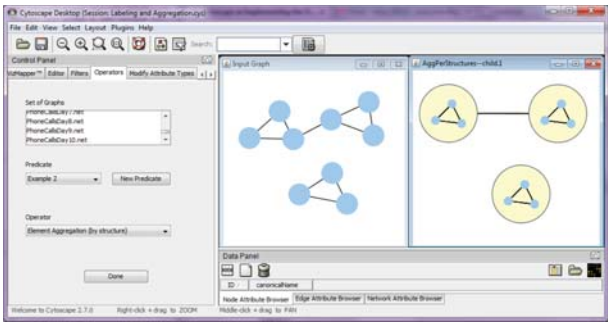

Figure 3.23: A Cytoscape Aggregation Per Structures is performed on the input graph shown on the left.

3.10 Experiment

A central motivation for the graph algebra is to support reproducibility of analysis. In this section, we evaluate this experimentally with a user study. First we define reproducibility and how it can be measured given an experimental analysis.

3.10.1 The Reproducibility Metric

We measure reproducibility by comparing the similarity between the $target$ and the $copy$ graph. The target graph is created when analyst A performs manipulations on the original graph, which we call the $start\ graph$. The recorded manipulations of analyst A is the $analysis$ used to create the target graph.

When analyst B is given A's analysis and the $start\ graph$, the graph she produces by following the instructions is the $copy\ graph$. Since she cannot see the $target$ graph, her copy may have errors depending on how well she followed the instructions and how well the analysis instructions are written. Therefore, reproducibility reflects the ease for other analysts to repeat an analysis on the same or different data sets.

We calculate reproducibility with the following metric:

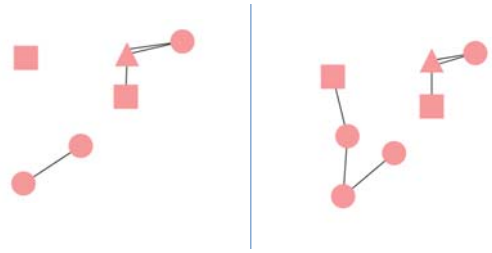

Figure 3.24: The graph on the left is what the final graph after the analysis should be. The graph on the right shows one of the graphs produced during the user study.

$$Reproducibility = \begin{cases} \frac{T-M}{T} & \text{No extra elements} \\ 0 & \text{Otherwise} \end{cases} \tag{3.1}$$

where,

$$T = \text{num of elements in the target graph} \tag{3.2}$$

$$M = \text{num of missing elements in copy graph} \tag{3.3}$$

We calculate the total number of elements in the target graph by summing the cardinality of the node and edge sets. The reproducibility metric is computed per graph. A higher value is a better reproducibility. Figure 3.24 shows a $target$ and $copy$ graph. The copy has 3 extra graph elements, namely, the circle and edges between the square and the lower left circle. The reproducibility is equal to 0.

3.10.2 Procedure

We test the reproducibility of an analysis reported in plain English versus in the algebra. Since there is not a universal systematic language that defines the basic graph data manipulation actions in visual analytics, English is usually used to share results. So, we compare sharing an analysis in English versus the Visual Analytic Graph Algebra.

The user study has two phases. In the first phase we show the start and summarized graph to 10 different users. All users are graduate students at the University of Michigan.

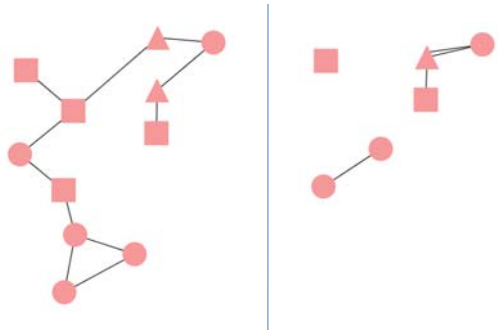

Figure 3.25: Starting from the graph on the left, the graph on the right is created, the summarized graph.

Figure 3.25 shows the two graphs. The tasks involve only element-level operators. Half of these users' task is to write in English the set of transformations to the start graph to create the target graph. This set of transformations is an English $analysis$. The other users are given algebra operator training. Their task is to create an algebraic $analysis$ for the target graph.

After phase one, we have 5 analyses written in English and 5 analyses written in the visual analytic algebra. In the second phase, we give one of these analyses and the start graph to 10 different users. Their task is to read the $analysis$ and perform its transformations on the start graph. The users who are given an algebra analysis are trained on the algebra operators. We compute the reproducibility metric on the replicated graphs.

3.10.3 Results and Analysis

The study was performed using pen and paper. We chose a graph with less than 10 nodes to make drawing and transforming the graph realistic for users. After experimenting with several different graphs, we selected the graph pair shown in Figure 3.25. We wanted users to quickly spot differences between the graphs without writing extremely naive analyses.

The graph algebra will not likely be used directly by analysts. It is a universal language between tools and methods that can be mapped to high level tool operations. For example,

in Cytoscape, creating a predicate is a drag-and-click process versus writing a complete algebraic predicate tuple. Since the algebra implementation in Cytoscape is within its early prototype stage, we want to separate noise from the users' system experiences from testing whether they are empowered or overwhelmed by the algebra. So we bypass system tools in the user study to test the algebra with pen and paper. In practice, the tool designer would need to be aware of the algebra and how the tool operations map to the algebra operations.

The following is an example of an English analysis written: "Start from the upper left corner, delete the square and move diagonally down and delete the circle and square and the circle again. Then take out the upper colored-in triangle and an edge from the triangle to the circle." The English analyses utilized node placement and deletions in the directions. The algebra analyses were more specific: creating attributes, selections, aggregations. Users were trained on the algebra for 15 minutes. They were all familiar with graphs and had previously taken a programming course.

English	Algebra
0.7	1
0.7	1
0.5	0.6
0.5	0.6
0	0

Table 3.1: This table shows the copy graphs' reproducibility sorted from best to worst.

We computed the $reproducibility$ for the copy graphs, shown in Table 3.1. The algebra has higher reproducibility in the copy graphs. Based on the p-value = 0.02573 from two sample t-test, the algebra group has statistically significant higher average score than the English group. These results are reasonable because the algebra provides a precise language for the analysts.

3.11 Visual Analysis of High-Throughput Biological Datasets

Hence, there is a growing need for visual analytics for biological networks (graphs) that this paper provides. The scientific questions under consideration, as well as the data available to address it can be fairly diverse and complex. Therefore, numerous ways exist that one can proceed with the analysis of such data. We present an algebra geared towards visual exploration and analysis of biological data with network structure that allow researchers a systematic and rigorous approach of the underlying problem at hand. The algebra consists of several data mining operators, coupled with the corresponding visualization operators. We also introduce some novel data structures, relevant to high-throughput biological data. The algebra is designed to interact with Cytoscape, a popular visual exploration tool in the computational biology community.

3.11.1 Biological Study Background

There are two biological studies that drive the visualization methodologies that we present. The first study [49] is a prostate cancer progression one. It uses high-throughput methods to collect expression levels of metabolites from a group of patients. The patients are divided into three categories: those with clinically localized tumors, those that the tumor has metastasized, and those with benign ones. Based on the expression levels, it characterizes the roles of the metabolites in prostate cancer progression. The statistical tests employed for assessing significant activity are chi-squared tests for assessing class-specific metabolite patterns of present and undetected measurements, two-tailed Wilcoxon rank sum tests for two-sample test and Kruskal Wallis for three-way comparisons. The visualization tools biologists routinely use, as is the case in this study are: heat-map of expression levels for all the metabolites and the patients, and aggregation of the pathways in relation to findings from other related studies based [19]. The main scientific finding of

the study is that sarcosine, an intermediate and byproduct in glycine synthesis and degradation, plays an important role in the disease's progression, since it attenuates prostate cancer invasion.

The second biological study [45] also is a prostate cancer progression study on the three stages of cancer mentioned above. It captures possible proteomic alterations in human prostate cancer progression. Then based on the set of measured proteins, they are able to identify corresponding genes from other biological databases. By utilizing both proteomic and gene expression information, the study identifies consistent patterns that can serve as predictors of clinical outcome in prostate cancer. Similar types of statistical tests are used to identify active genes.

In this example, we also take an additional step in the analysis of these data sets. Specifically, we link gene and metabolic pathways (identified in the KEGG database) to gain insight into the molecular mechanisms underlying the disease.

3.11.2 Data Sets and Tools

Data Sets

In addition to the data from the two studies described above, our data set comes from well-known biological data bases. We obtain the pathway networks from the Kyoto Encylcopedia of Genes and Genomes (KEGG) [36, 38, 37]. The network is heterogenous because the nodes can be either type gene, compound, or a collection of genes. The collection of genes work together as a complex. There are 201 pathways in KEGG. Each pathway is a biological process; for example, the Galactose metabolism pathway describes the reaction on converting galactose to glucose 6-phosphate in liver.

We combine the pathways by finding overlapping nodes. Since some nodes are a collection of genes, in this case, we consider two nodes overlapping between different pathways when all genes are present in both nodes. This is the strict method to find overlaps.

From supplementary information in [49], we also know the significant metabolites in this dataset with respect to prostate cancer progression.

For the gene expression data set [45], we have the gene expression levels along their significant values from each of the 13 patients on over 54,000 affymetrix IDs. There are 4 patients coming from benign and metastatic groups and 5 patients coming from clinically localized prostate cancer group. The significant value (p-value) on each gene provides us with information on how precisely the expression is collected. Based on the gene symbols, we are able to attach the expression levels with p-value less than 0.6 to 4329 genes in all 201 KEGG pathways. If there are multiple affymetrix IDs under the same gene symbol, we will pick the expression levels with the smallest p-value.

We apply log 2 transformation to stabilize the variances in the expression data set. In order to eliminate the heterogeneity from individuals, we normalize the transformed data with respect to pool samples based on the quantile normalization method proposed in [11] within each cancer group.

Figure 3.26 shows a heat-map of preprocessed gene expressions across different cancer classes. Due to the very large number of genes involved, this image provides little information on the relationship between gene expression and cancer stage.

Tools

Cytoscape [16] is an open source visual analysis tool for biological interaction networks. One of its advantages is the large user base who also contribute to increasing the tool's functionality. Cytoscape supports the implementation of these contributions and deploys them as plugins to the tool. Various plugins were employed including: the NetworkAnalyzer and Enhanced Search Plugin. The NetworkAnalyzer plugin calculates basic statistics about the network and saves them in their appropriate nodes or edges. The Enhanced Search plugin allows simple queries on node predicates in the graph. There are

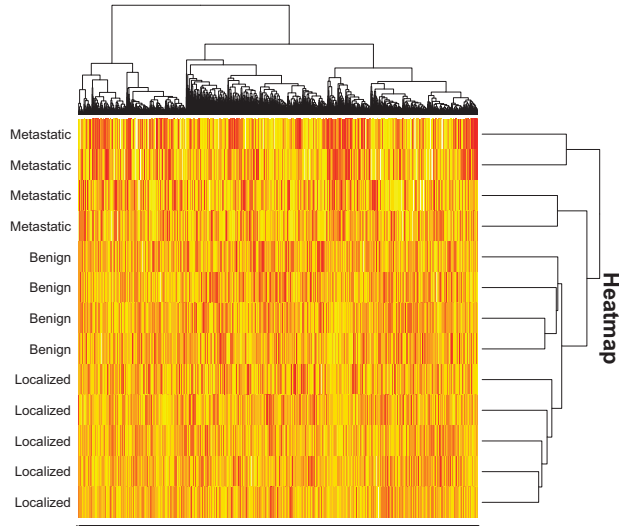

Figure 3.26: Heat-map for 4329 genes. The orange is positively expressed and yellow is negatively expressed.

many ways to display a network in the window. Cytoscape can also display the nodes and edges in different colors, shapes, and sizes according to selected attribute distributions. Other useful features of Cytoscape include the history panel which displays all the graphs during the current session.

R is an open source statistical analysis tool. Bioconductor [17] comprising of 352 packages written in the R language, is a collection of statistical techniques routinely used in the analysis of high-throughput biological data. Some examples of useful packages used in our study include the *multtest* package, for performing permutation tests for assess differences between the benign, localized and metastatic groups of patients, the *Limma* Package that fits linear regression models to expression data, the *gplot* package that creates heat maps (see Figure 3.26), etc. It should be noted that these packages were designed to address a separate computational/visualization task and do not work in an integrated manner. The user needs to write a lot of additional code in order to have a comprehensive discovery process. Our algebra translated in Java and R code, allows users to design such a process.

3.11.3 Analysis

Our goal is to support biologists trying to make sense of large data sets collected through high-throughput experiments. We begin with massive amounts of data in gene pathways and metabolite pathways. We also have expression level data. Figure 3.27 shows all the gene pathways in a combined network. As we can see this is one of the staple hairballs in the biological world. The intrinsic attributes of each of the gene nodes are: Affymetrix ID, hsa gene encode, Gene Symbols, Chemical structure and a multi-valued vector contains expression level from 13 patients. Each of the metrics correspond to a function created into a computed attribute. The graph is displayed in a force directed layout. Very little information can be obtained from this graph. But nonetheless, this is the starting point for high-throughput analysis of biological data. We also include the gene hairball for a complete road map of our analysis.

The diameter of the network is 21, the node count is 2330, and the edge count is 6128. Note that although there are 2330 nodes in the gene pathway in Figure 3.27 there are over 4000 genes in the heatmap in Figure 3.26. The reason for this discrepancy is that some nodes in the gene network are comprised of a complex of genes that work together. Also, of the 4000 genes, several may be orthologs [49].

Visual Reduction Analysis

We continue our case study analysis focusing on the biological studies' work outline. In our analysis we do not claim to make new biological discoveries. Instead, we follow the path of analysis from the presented studies [49] and [45]. We also provide a way to link the two studies. We will conduct our analysis simultaneously through the visual algebra.

Our first task is to understand the relationship between the pathways and the enrichment score. We will have to make several manipulations and compute metrics to complete this

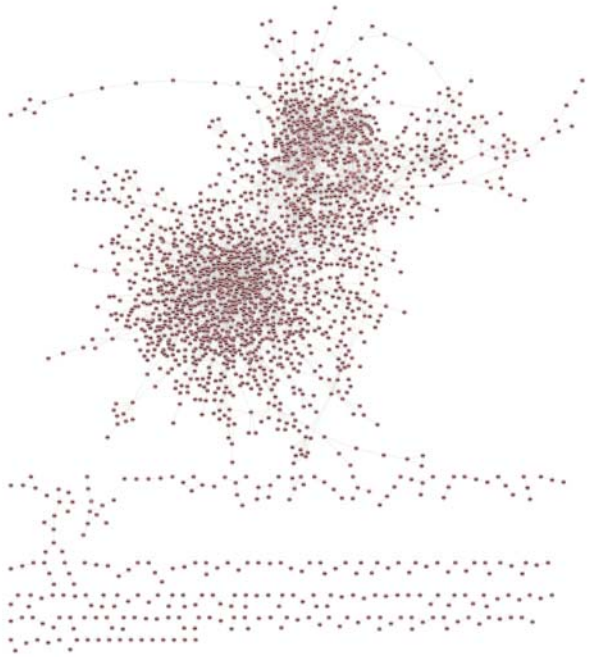

Figure 3.27: This figure shows the entire gene pathway network.

task given our starting point. The enrichment score of a pathway is a ratio of the number of significant genes, under a predefined hypothesis, over the total number of genes within the given pathway. In this application, the significant genes are defined as genes with significant expression shifts among the three stages of prostate cancer. We select significant genes based on adjusted p-values from t tests at 5 percent level of significance. Both adjusted p-value and enrichment score correspond to computed attributes in our algebra. The reason biologists use this score is because it reveals the importance of a pathway which is related to a specific phenotype of interest. The enrichment score is an attribute of a pathway, but at the current granularity of the graph, we have only genes.

Each gene has a set of pathways it can belong to. This can be represented as a structured

intrinsic attribute on a gene. Specifically, we have the structured attribute as a list of possible pathways that gene X is present in. One way to store the enrichment score value is to store it at the gene granularity of the graph as another structured attribute which is a vector with the enrichment score of each pathway. This structured attribute can be placed at the network level or replicated at each gene node. Recall that our task is to prepare visualizations that can understand the relation between pathways and enrichment score. If we create the computed structured pathway enrichment score on the network level we will not be able to visualize the relation information because edges can be seen at the node level. On the other hand, as we saw in Figure 3.27, the gene hairball is too complicated, even if we replicate the pathway enrichment score on each node, we will not be providing any visualization to understand the problem.

At this stage we must reduce the graph. The natural choice is to group the genes by pathway. This way we can visualize the relations between pathways. So we create essentially a new type of node, representing a pathway, which is comprised of the genes within that pathway. We place an edge between pathways if there exists a mapping node in the KEGG database to another pathway or if there is an overlapping gene between the pathways. This operation corresponds to the element aggregation where the predicate is a pathway. The attributes on this new pathway node will be: enrichment score, number of genes, a structured attribute vector of the genes, and the pathway name. Recall that there are 201 pathways in the KEGG dataset.

Figure 3.28 shows the pathway graph in a force directed layout which is highly connected. The color is a gradient on red corresponding to the enrichment value of the pathway. The lighter the color, the higher the enrichment value. Even with the aggregation at Figure 3.28 it is difficult to see any interesting structure. We can reduce the graph more by thresholding at different levels of enrichment score to understand the problem. This

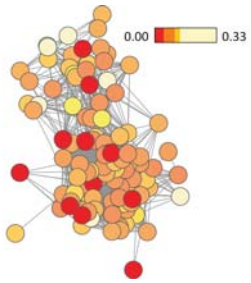

Figure 3.28: Genes are aggregated by pathway and gene enrichment level is shown. All of the 201 pathways are present.

corresponds to the element selection operator in the algebra: we use a predicate on the enrichment value, $\alpha = \{V_1, \mathbb{X}_V = \{X_{v.type} = pathway, X_{v.enrichmennt} > \text{constant}\}\}$. Figure 3.29 and Figure 3.30 display the gene pathways with enrichment score over 0.066 and 0.1, respectively.

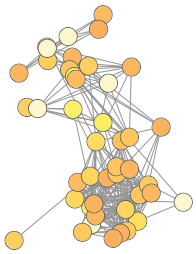

Figure 3.29: Genes are aggregated by pathway and gene enrichment level is shown. Pathways with enrichment score above 0.066 are shown. The same gradient as in Figure 3.28 is used.

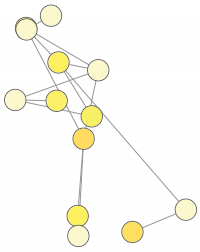

Figure 3.30: Genes are aggregated by pathway and gene enrichment level is shown. Pathways with enrichment score above 0.1 are shown. The same gradient as in Figure 3.28 is used.

Figure 3.32 shows the pathway in Figure 3.30 from a different point of view. Based on the pathway information on the node level, we compute the gene expression for each path-

way by averaging all gene expressions. The color is a gradient corresponding to positive and negative expression level of the pathway. For each pathway, we have average gene expression for metastatic (left), clinically localized (middle) and benign (right) prostate cancer groups. It is easy to see that many connected pathways have similar expression levels for all three cancer groups.

Through this process we have gone from Figure 3.27 to Figure 3.30 which is able to provide insight on gene networks based on statistical testing, but our steps have been completed with the proposed algebra. This makes future replication of this reduction analysis process possible.

Heterogenous Network Joint Analysis

The next task is to do a joint analysis of the metabolic and gene networks. This is a relatively new territory in biological studies, especially at the high-throughput scale. As we saw in the first task, staying within a homogenous set of molecules, there are still too many nodes to see meaning. We used reduction and summarization methods to understand the data. The algebra is able to handle heterogenous node types and perform the same reduction operators. In the next task we will prepare a visualization capable of doing a joint analysis on metabolites and genes. This visualization will provide biologists to see any relation between significantly expressed genes and significantly expressed metabolites. One possible benefit for biologists to see a correlation is substituting one molecular data set for another, depending on reliability of the data set.

To begin this task, we start again at Figure 3.27. But now we include metabolites from KEGG into this network. This again forms a hairball. Because of the heterogenous node types, this causes a heterogenous edge types as well. Edges between genes versus edges between genes and metabolites have different meanings to biologists. From the study [49], we know that sarcosine plays a important role in prostate cancer progression. Therefore,

we are going to start our analysis by selecting the pathways which have the sarcosine metabolite. In our algebra this correspond to the following operators:

- Element selection with predicate pathway name, for each of the 201 pathway names. This step creates 201 graphs, one for each pathway.
- Set selection on the 201 pathway graphs with predicate molecular name Sarcosine. This step returns the graphs where Sarcosine exists within the pathway.

There are only two pathways in KEGG that this is true: Arginine and proline metabolism pathway and Glycine, serine and threonine metabolism pathway.

We want to make the graph visually capable of understanding the three attributes. The three attributes of interest are the following:

- Node type: which can be either metabolite or gene
- Significantly expressed in the study: which can be either true or not.
- Pathway: this is a vector of the pathways the gene is found.

Figure 3.31 presents a visualization that allows an analyst to perform the task of a joint metabolite and gene analysis on significance level of the molecules in the KEGG pathways. We use Cytoscape to display a different shape for the node types, and different color for the significance values, we also specially denote the Sarcosine molecule which is of particular interest in the [49] study with a bright green color. Then we use a circular layout by pathway attribute. The circle on the left and right of the Figure 3.31 is for the Arginine and proline metabolism and Glycine, serine and threonine metabolism pathways, respectively. The middle circle are the nodes that overlap between the two pathways.

We minimize the complexity in color usage on the attributes for each node type by using a gradient on the same color. Finally, we increase the width of the adjacent edges to

Sarcosine and a significant gene. This image shows that Sarcosine interacts with other significant metabolites. Moreover, significant metabolites also connect to gene with notable expression changes.

The joint analysis visualization shown in Figure 3.31 can be transposed to other biological studies. For example, studies on proteins and metabolites can use a similar visualization structure. Currently there is still a lot of missing information in KEGG [36, 38, 37], as more information and more expression data on different types of studies become available, visualizations, as shown in Figure 3.31, will provide more insight. New visualizations can inspire new biological studies and hypotheses. Also, new biological data drives new ways to visualize data.

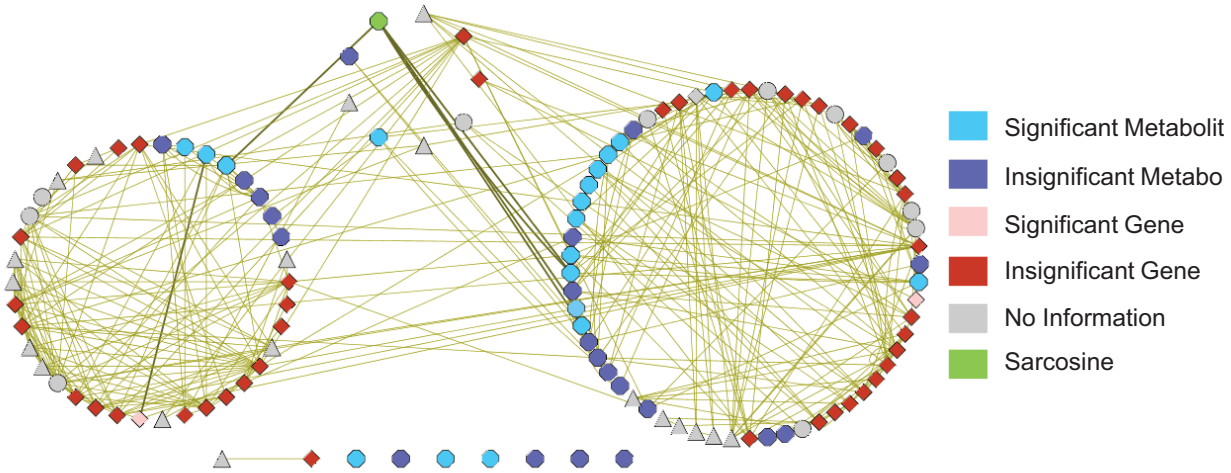

Figure 3.31: This picture shows the connection between the metabolites and the genes network for the two pathways of Sarcosine. Node color indicates significance in expression level shifts for three stage of prostate cancer.

3.12 Optimization

Visual Analytic tools are interactive. Each exploration step needs to be fast enough so that the analyst is not interrupted. An un-optimized complicated workflow may cause an

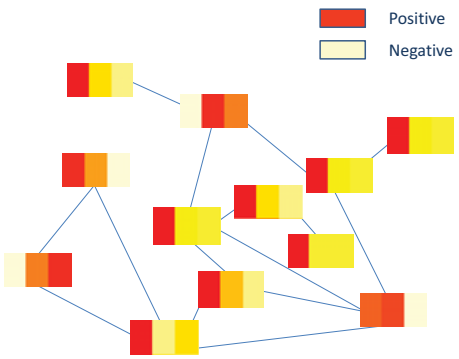

Figure 3.32: Average gene expression levels for pathways with enrichment score over 0.1. Red stands for positive gene expression and yellow stands for negative expression. For each node, pathway, average gene expressions are listed by metastatic, clinically localized and benign prostate cancers.

exploration on a data set that takes longer due to too many extraneous steps, time intensive operations, or complicated designed predicates.

Optimization in interactive workflows, in which a visualization might be expected after each tool function is performed, eliminates some ways we can optimize. In a batch optimization, we can reorder all the operators as long as the final operation produces the same outcome.

Batch optimization actually becomes helpful in many exploration situations where an analyst has to repeat a similar workflow. They may want to modify predicates or use a different data set. If a workflow has been saved for a target visualization, we can optimize the workflow for the new conditions. We assume an analyst saves a workflow by saving a visualization they have created. We use Cytoscape's Visual Analytic Algebra Plugin. Each time a visualization has been exported, the algebra log to create the picture is saved. So when we perform batch optimizations, we perform the optimization over the saved log. In this chapter we discuss techniques to optimize batch workflows and how the optimization may improve exploration. In a batch of operations we can reorder all operators as long as the final operation produces the same outcome.

The motivation behind this type of optimization is that an analyst conducts a very similar workflow daily on new data sets. To streamline this process we propose optimization techniques on the workflow. Since it is unlikely that each repeated run of the workflow is identical, we show techniques for an analyst to modify parameters of a previous run. Therefore, our contributions are two part. First we identify heuristic based optimization techniques to reduce the number of graph isomorphisms to be computed. Next, we present through a case study how we envision to interact with a workflow optimization technique. The techniques to reduce the number of graph isomorphisms identify the following inefficiencies in exploration. The advantage of a batch based optimization is that we know what the target visualization is and we have a set of operators which will lead us to the target. So, if an analyst previously took a circuitous route to a target visualization, we can bypass unimportant steps.

Our objective in optimization is to reduce the number of graph isomorphism to perform. Most operators in the algebra use the Graph Matching function, which is responsible for finding isomorphs (which we call witnesses) of the predicate query graph in each input graph. The reason reducing the number of calls to the Graph Matching function is important to optimization is because graph isomorphism is an NP-complete problem. This complexity makes the Graph Matching function a time consuming part of the graph visual analytic algebra. We can also use heuristics and optimization techniques for the Graph Matching function itself. Usually these methods either use graph matching approximations or indexing on the predicate and input graph. Graph indexing is a popular approach because with a clever index, many candidate nodes in the input graph can be filtered (i.e. they do not match the predicate nodes characteristic) before a graph isomorphism is performed. We discuss current graph matching optimization techniques in the related work subsection.

There are other costs in using a graph algebra, for example, a cost associated with the times a user takes to write a query, choose which operator to use, etc... We leave these areas for future work.

We explore the visual analytic graph algebra rewrite rules which will result in fewer Graph Matching function calls. We describe different rewrite possibilities for the different components of the graph visual analytic algebra: predicates and operators.

Since the visual analytic algebra operates on sets of predicates (i.e. sets of graphs) and returns a set of graphs, we look into ways to optimize the input predicates so we can perform fewer graph isomorphisms.

For example, there could be redundant predicates in the input set. If we can find duplicate predicates, then we can eliminate future expensive searches against the potentially many and larger input graphs. In the following sections, we elaborate on these ideas and other rewrite rules. We mix general optimization methods with rewrite rules in our discussion.

The remainder of this section is organized as follows. First, we present related work specific to this chapter. Then, we present optimization rules and methods for the optimization engine. Finally, we show how a batch workflow can be re-run with new parameters and datasets in Cytoscape.

3.12.1 Optimization Engine

The main objective in optimization is to reduce the number of graph isomorphisms to perform. The selection and aggregation operators use the graph matching function to identify witnesses. Each witness identification is a graph isomorphism. As presented in the algebra chapter, the inputs to the operators are in the form of a predicate. One way a workflow might be slowed down is by operating on complicated predicates. Example complicated predicates which can be optimized include redundant predicates or predicates

with identical substructures. We present a set of optimizations for predicates. Next, we identify ways to optimize an individual operator. Again, by "optimize" we mean to reduce the number of graph isomorphisms necessary to compute the output results. Given the Cytoscape implementation of the algebra, after each operation a visual operator can be applied. Therefore, optimizations of composition of operators is not possible in the Cytoscape instance of the visual analytic algebra. In our presentation of optimization, we examine predicate and individual operator optimizations. However, we briefly discuss optimizations for a composition of operators (i.e. two consecutive element selections). Firstly, element aggregation is not commutative or associative because after each element aggregation operator the graph topology is changed. However for set aggregation and selection the topology of the graph is not changed. Graph elements might be eliminated or added, but the no new connections and nodes will emerge in the output which did not exist in the input graphs. The proof for this property is in the definition of the operators. Therefore, there is more room for optimization on a composition of set aggregation or selection (element or set) operators. For example, the operators can be re-ordered and their predicates can be merged.

When a predicate is designed through the NetMatch, it is saved as a predicate graph to choose as an input to operators through the operator panel. Some of these predicates might be very similar to each other. For example, they could differ in the attributes on the graph elements. For example, an analyst may have set of possible criminal rings to identify in the network. The criminal rings may differ by slightly different information on attributes. For example, two predicates might differ on the location attribute of a node. To keep the two hypotheses predicates distinct, the analyst could design two predicates. However, in the graph matching phase, it would be ineffective to repeat the graph isomorphism for both predicates when their structure is identical. So the first heuristic technique to reduce the

number of graph isomorphisms to perform is to reduce the number of distinct predicate structures.

We list some ways to reduce the number of predicates:

1. The most naive case is if there are duplicate predicates in the set of predicates. In this case we can map each predicate p_1, ... , p_n to a single arbitrary predicate p_x. We can use this predicates match list whenever any of p_1,... , p_n is used an input to an operator.

2. The next more complicated scenario is if two or more predicates have identical structure but their attributes have different conditions. In this case, the similar predicates all refer to the same predicate p_x, but p_x has no attribute conditions.

The above reductions rely on identifying similarity and duplication among the predicate lists in an analysis session. Secondly, they require a way for a predicate to reference another predicate. This referencing can be implemented by the graph matching operator. Before performing a match, it can check if the predicate refers to a previously used predicate. We define duplicate predicates as two or more predicates with identical structure and conditions on the attributes. Testing for identical structure requires graph isomorphism. However, we would be trading off graph isomorphisms between predicates and large data sets for isomorphisms between comparatively smaller graphs. Graph structural similarity is looser than duplicate predicates. If two predicates are similar, then they have identical structure, but the conditions on the attributes might be different.

Another way to reduce graph isomorphisms is to compare the predicate attribute conditions, specifically the graph level attribute conditions. If a predicates graph attribute conditions are not satisfied there is no need to examine the nodes and edges. We describe a heuristic method to create graph attribute conditions on a predicate, which may not have

had these conditions explicitly defined by the analyst. One way to create graph attribute conditions on a predicate is to push up node and edge attributes to the graph level, if possible. For example, if all the nodes have the same attribute value, this can become a graph attribute. If the input graph has defined this graph attribute value as well, then a comparison on the graph attribute level can be performed before examining the nodes and edges.

We discuss some heuristic techniques to optimize the individual operators. Element operators (selection or aggregation) can be optimized through current state of the art graph querying techniques. However, set operators operate on a predicate by identifying if any match occurs; therefore, to reduce the number of graph isomorphisms in a set operation, if we are given a set of predicates, we can choose the predicates which are least likely to occur. Again this is a heuristic technique which requires metrics to define whether a predicate's features are uncommon in the data sets study. For example, if a predicate structure has a degree distribution which is not represented in the input graph, then it is less likely there exists witnesses for this predicate in the input graph.

3.12.2 Case Study

The analyst may have built a visualization and, as a by-product of the Visual Analytic Plugin, produced a log of operations. When the session is loaded back to Cytoscape, its corresponding algebra log is loaded into the log panel of Cytoscape. The session represents a target visualization which was saved. It's corresponding log is the workflow used to build the target visualization. An analyst can use this workflow and prebuilt target visualization in a session, to continue work on new data sets by loading a new data set into Cytoscape. The analyst can select and highlight operators from the log. They can update the parameters of the log. For example, select a new data set, select a new operator, select a new predicate, and change these values directly in the algebra log. And then re-run the

log. Therefore, the log is a place for analysts to manually enter the queries in operator form. When operators are re-run through the query log, the optimization engine is used to improve the efficiency of the predicates and operators used. We show this process through the thesis overview example.

Scenario

We study the thesis overview example on the Cell Phone Challenge from VAST 2008 [4]. This challenge requires analysis on a set of 400 unique cell phone call records over a ten-day period to learn the Catalano social network structure. The data set includes 9834 phone records with the following fields: calling phone identifier, receiving phone identifier, date and time, duration, and the cell tower of the call origin. The purpose of the challenge is to identify the Catalano/Vidro social network at day ten and to characterize the social structure changes throughout the time period. The first part of the challenge requires identifying Ferdinando Catalano, Estaban Catalano, David Vidro, Juan Vidro, and Jorge Vidro. Along with the data, the challenge provides a lead that Ferdinando Catalano is identifier 200. Also Ferdinando calls his brother, Estaban, most frequently. Finally, we know that David Vidro coordinates high-level activities and communications within the network. The goal is to identify the criminal ring in Figure 3.33.

In this case study, we assume that the analyst has identified the criminal ring in the data set for days 1-7. They want to reproduce the analysis they conducted on days 1-7 for the data set on days 8-10. We show the workflow of the analysis in Figure 5.10.

The analysis for the target visualization, which identifies the criminal ring for days 1-7, is loaded onto Cytoscape, shown in Figure 3.35. The algebra log which corresponds to this target visualization is available in the algebra log panel. We summarize the key actions to reach the target visualization.

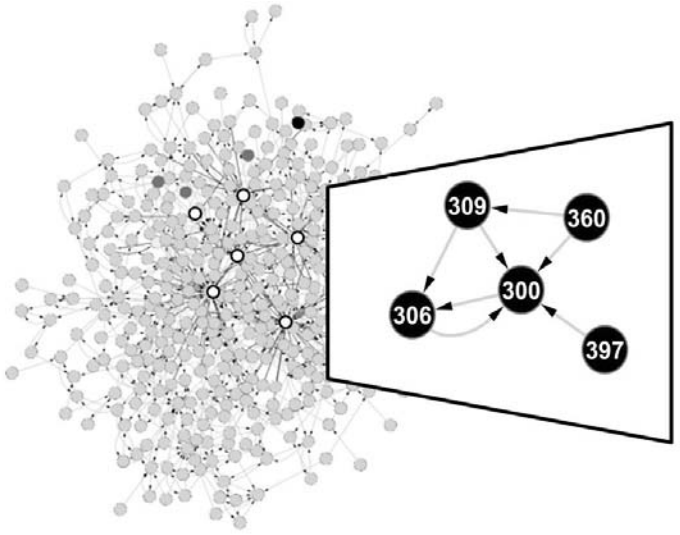

Figure 3.33: The criminal ring structure.

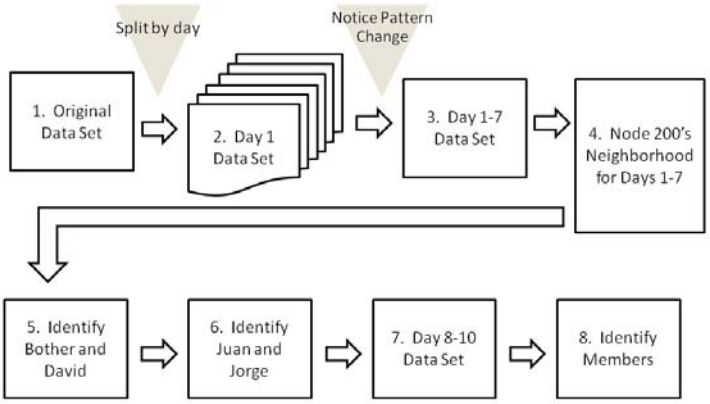

Figure 3.34: The workflow for analysis.

- Load in the data set: $D^0 = G = (V, E), X_E = (X^{date}, X^{duration})$
- Draw graph on Cytoscape: Tool specific visualization $\mathcal{V}$
- Split dataset into ten days: Element selection $D^i = \sigma^{selection}_{element,\alpha} D^0$ where D^i is day i and α is the predicate to represent day i (i.e. $X^{date} = i$)
- Draw graph on Cytoscape: Tool specific visualization $\mathcal{V}$
- Aggregate dataset on days 1-7: Set aggregation $\phi_{set}(D^{1-7})$
- Select node 200 and its induced subgraph: Element selection $D^{200} = \sigma^{selection}_{element,\beta} D^{1-7}$ where β is the predicate to represent node 200 and its neighbors.
- Draw graph on Cytoscape: Tool specific visualization $\mathcal{V}$

Next the analyst would like to produce the same visualization for days 8-10. To re-run the same workflow, the analyst must update the datasets and the parameters to the operators. This can be done by highlighting and copy and pasting the previous log operations, updating the parameters, and finally running the operators. The optimization methods described in this chapter can be applied on the operators and predicates when the workflow is run.

3.13 Conclusions

In this chapter, we introduce a framework and the corresponding algebra for visual analysis of graph based data sets. We provide a detailed description of its operators and illustrated its use through an implementation in Cytoscape.

We also presented techniques to optimize operators and predicates to reduce the number of graph matching function calls. We also demonstrated how a workflow can be re-run in Cytoscape.

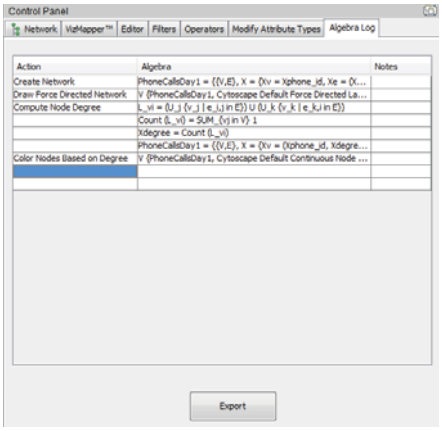

Figure 3.35: An Algebra Log.

CHAPTER IV

Green Trellis

4.1 Introduction

Interesting patterns in graphs can take many forms, and so can graph analytics applications of interest. A user exploring a phone call network, where nodes represent callers and edges represent calls, might know which callers and their characteristics she wants to study. Or she might not have any information on the callers, just the knowledge that there is some suspicious activity and it is her task to find a possible lead. To perform her task, the user must compare groups of nodes, assess major trends, and spot nodes with unusual behaviors. An effective tool for graph data exploration must be able to support the user in this broad variety of circumstances.

Over the past decades, researches invented sophisticated visualization tools to support graph analysis in a variety of circumstances mentioned above. We describe these works in the Related Work Section (Section 2). In conjunction, there has been effort to develop a taxonomy of graph analysis tasks so that tool developers have an aim in their methodology. Given these advancements, we feel there is a neglected fusion in the graph community, which can improve current tools. The theoretical graph community identified several metrics to model complex graph structures. Popular ones, such as degree and clustering coefficient, are widely used in graph software. But several are untapped in graph tools, for

example, cyclic coefficient and neighborhood metrics. Furthermore, most tools focus on metrics that describe either a node given its immediate neighbors or the entire network. A relatively unexplored idea is to summarize interesting features on a continuous growing distance from each node.

To pull these theoretical concepts into an exploratory tool, we utilize graph signatures. A graph signature is a numeric vector that represents topological characteristics of a node. Visualizing these signatures on a low-dimensional scatterplot reveals local and global graph features not visible on a force-directed plot. There are many different ways to define a graph signature, with each definition bringing to light selected graph properties. In this chapter, we present a visual graph exploration tool that conveniently juxtaposes a variety of novel graph signature techniques, permitting the user to go back and forth between them with ease, in addition to the traditional force-directed graph layout. We call this tool GreenTrellis. We demonstrate through example the effectiveness of GreenTrellis in helping analysts find patterns of interest. With GreenTrellis, an analyst can examine different communities of nodes, compare and contrast graphs, zoom into structures of interests, and control the locality of the scatterplots.

We present ways to expand the topological characteristics represented by a graph signature. These different signatures include the following characteristics: neighborhood connectivity, cyclic properties, a nodes tendency to share neighbors, and clustering coefficients. The metrics we select are independent of graph specific attributes. We show how to encapsulate any graph metric as a signature in GreenTrellis.

To show that GreenTrellis can be used in graph exploration, we provide procedural solutions to selected tasks from compiled taxonomies [22]. However, the main usability challenge we face is to fuse graph theory into a tool for analysts without theory expertise to intuitively apply within the context of their problem. We test how effective GreenTrellis

is for an analyst to explore the data through usability studies. We examine three topics: visually performing subgraph matching with advanced graph signatures, comparing large graph structures by varying signature locality, and characterizing a graph with the appropriate signature. In summary, we make the following contributions:

- A locality-driven multi-modal scalable graph summarizing tool
- Fusing advanced graph topology representations into a visual analytic system to create visual indexes over data
- Procedures to solve various graph tasks taken from compiled taxonomies
- Usability study to examine how effective GreenTrellis is in various analytical situations

The remainder of this chapter is organized as follows. First, we present related work specific to this chapter. Next, we present the GreenTrellis tool by describing each of its features and how they come together to solve various graph tasks. Next, we describe the theory and computation behind the new graph signature designs. Then, we present a usability study with multiple datasets. Finally, we conclude and describe opportunities for future work.

4.2 Related Work

Wong et al. [60] introduced the topological breadth-first search (BFS) graph signature implemented in GreenTrellis. To construct a graph signature at a node, the algorithm traverses the graph using BFS to collect local topology information. The algorithm sums up the graph connections according to the distance from the root of the BFS hierarchy. The depth of the search depends on the analysts interests in how the nodes relate to proximate or distant nodes. The next step uses multidimensional scaling to project these signatures

onto a low-dimensional scatterplot in which similarities of node signatures are approximated by distances between nodes.

The strength of this signature is ease of implementation, speed of computation, and ability of analysts to intuitively understand the scatterplots. However, relying on only one signature neglects interesting graph features. In GreenTrellis, we apply other graph metrics and characteristics into the graph signature form to display more information about the graph. Newman [42] provides a survey of different graph features we utilize. The database community uses graph signatures for communication network analysis [18]. A signature resembles an index used in relational databases that filter and manage large amounts of data. For example, [23] computes a topological index for each node to perform faster matching and comparing of subgraphs. In our work, we visualize these signatures in GreenTrellis to help analysts interactively process queries.

4.3 GreenTrellis

In this section, we present the design goals which inspired the different features and data structures in GreenTrellis. We describe the implemented tool features which help achieve each design goal. Next, we provide procedures to solve a selected range of exploratory problems from compiled graph task taxonomies.

4.3.1 Design Choices and Methodology

GreenTrellis, shown in Figure 4.1, is a scalable graph visual analytic tool for analysts to pose, explore, and discover interesting features in graphs at any granularity. The left panel is an enlarged view of the graph in a force-directed layout. On the right is a matrix of graph signature scatter plots. All plots are linked together with brushing. Analysis is driven by selecting nodes in one plot and visualizing how this selection affects which nodes are highlighted in the other plots. We centered GreenTrellis design to maximize the

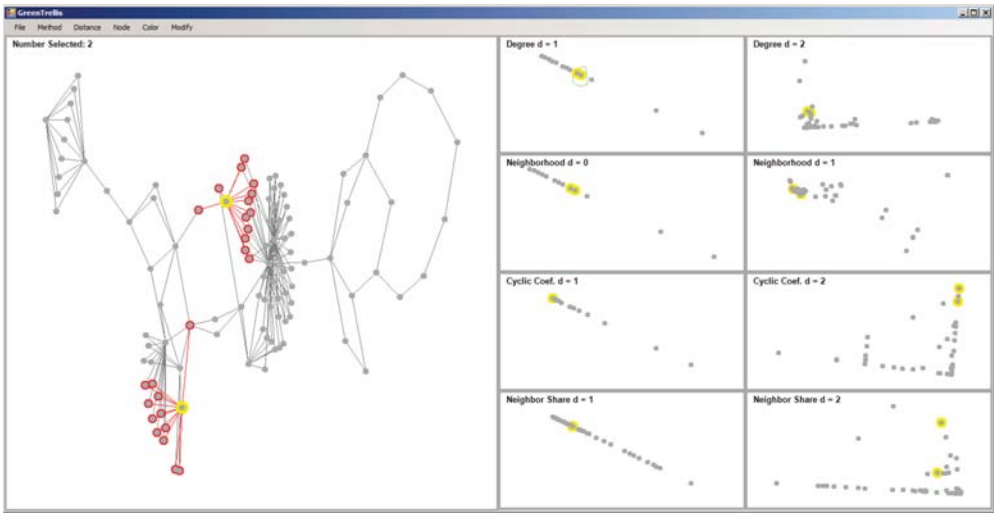

Figure 4.1: GreenTrellis links several different views of a graph. The left panel shows the traditional force directed graph and the matrix of scatter plot show various signatures at different distances.

following goals: (1) the tool should have functions which allow the analyst to understand the graphs structure, (2) the analyst should be able to identify features at varying graph locality, and (3) the analysts should be able to customize the tool to their specific tasks. In this section, we elaborate on our design goals and their supporting features in GreenTrellis.

Graph Structure

It is conceivable to design a sophisticated visualization which captures multiple characteristics of a graph with shapes, colors, and other visualization techniques. But a complex visualization might not necessarily maximize an analysts understanding of the graph structure. In this subsection, we list the features which allow an analyst to navigate through GreenTrellis to understand a graphs structure.

Graph Signature Scatterplots: These panels show node groupings according to different structural features. The user can customize which signatures to display. The process to produce a scatterplot is as follows: First, we compute a graph signature vector for each node. Then, we display these vectors using multi-dimensional scaling onto scatterplots. In (Section), we discuss the theory behind the graph signature computation. When two nodes

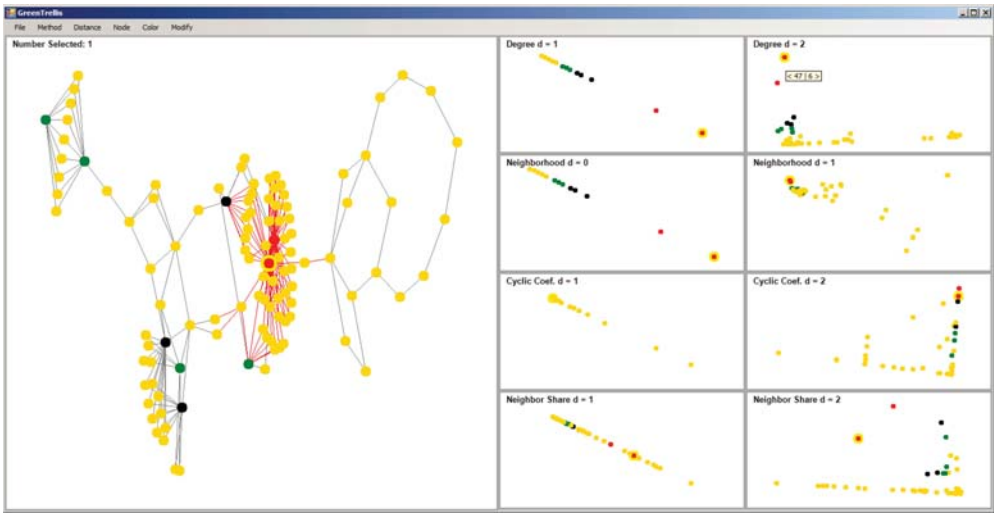

Figure 4.2: Groups of nodes have been colored to aid exploration. Right-clicking a node displays its signature vector.

have the same or similar vectors, they are drawn over or near each other; thus, creating a grouping effect.

Node-Link Panel: This panel displays the graph in the classic force-directed layout. A node can be dragged to see its immediate neighbors. This panel is linked to the scatterplot panels through simultaneous brushing effects during node selections. We can zoom and pan around the graph. The node-link panel is a classic and widely used view in graph analysis. An analyst can zoom into a region to see the node connections and understand the structure.

Color Graph Regions: On any panel (node-link or the scatterplots), the nodes can be re-colored and the new color will be updated on all panels. Coloring is a way to visually track the nodes based on the information learned during analysis. Figure 4.2 shows the effect of coloring groups of nodes.

Extract and Combine: This function extracts the selected subgraph and redraws it in a new color next to the parent graph. All of the panels will be updated. By extracting and combining a subgraph next to its parent, an analyst can compare the unique subgraph

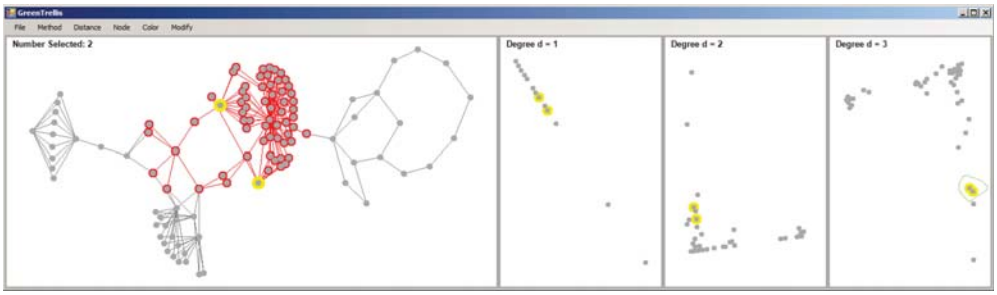

Figure 4.3: Since a group of nodes at distance 3 have been selected, the force directed plot highlights the graph induced a distance 3 away from the selected nodes.

features against the parents structure.

Annotations: Right clicking on nodes and different tool options triggers a tool-tip description of the respective element. For example, in the scatterplot panel, a right-click on a node displays the signature vector, shown in Figure 4.2. These annotations help the analyst to understanding the theoretical basis behind the visual information.

Locality

One of the contributions of GreenTrellis is that it can show graph features at varying distances away from nodes. This feature fulfills our second design goal of creating a tool in which an analyst can identify features at varying graph localities. The reason this goal is necessary in a graph tool can be seen when studying large graphs. A graphs structural features vary at different localities. For example, at a farther distance from the node, differences between the induced subgraphs may become more obvious. In the User Study Section (Section) we explore this phenomenon. The following are the features implemented in GreenTrellis to support locality:

Locality on Signatures: An analyst is able to vary the distance the graph signature scatterplot represents. The columns in the signature panels show increasing distances. These panels help the user compare the structures at varying distances from the node.

Distance Highlighting: Depending on where a selection is made, the distance that panel

represents is shown on the node-link panel through red-outlining of the nodes and edges within that distance, shown in Figure 4.3. This feature helps show the analyst what distance the selected graph signature scatterplot represents.

Customization

The third design goal is tool customization. GreenTrellis is not locked into one story of the data, but the analyst can discover the story behind the data. We offer tool customization options for the analyst to pose questions and control the direction of the study. For example, is the graph task to compare graphs, or, to characterize a graph? The analyst can define the graph task because of the customization options. The following are the customization features implemented in GreenTrellis:

Graph Sketching: The analyst can sketch a graph in the node-link panel and compare it with the existing graphs structures by using the linked scatterplots. The sketched graphs features will appear on the scatterplots. The analyst can also modify the current graph by adding and deleting nodes and edges to see its effect on the scatterplots.

Draw New Scatterplots: GreenTrellis offers many different graph signature methods to characterize features. An analyst can select different methods to display on the scatterplots. We provide descriptions for signatures and distances through the tool-tip.

Set Logic Operations: The analyst can apply different graph operations to modify sets of graphs. We provide union, intersection, and difference operations on graphs to produce a new graph. These functions are for advanced analysts who prefer manipulating the graphs with set logic.

4.3.2 Procedures towards Graph Task Taxonomy Solutions

Graph analysis covers many tasks, for example, matching graphs, characterizing structures, finding outliers, comparing two graphs, or mining the attribute data, etc. These

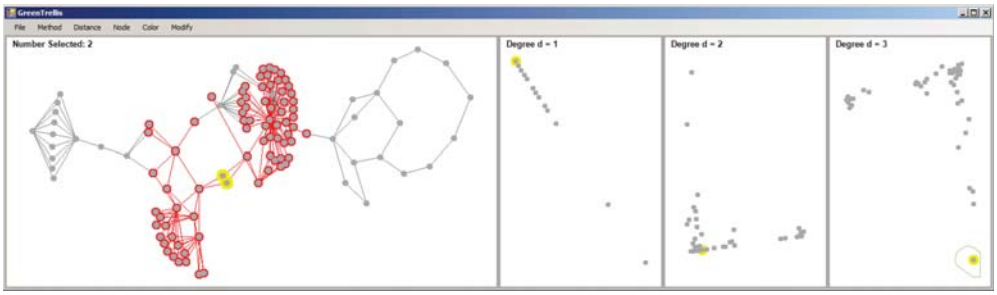

Figure 4.4: Outlier nodes are selected at distance 3.

questions usually do not start in a graph task language, either. For example, a social scientist studying trade agreements between merchants in historical villages may ask, What communication protocol did the most successful merchants follow? This question translates into the network formed question, What subgraph structure are nodes with high degree commonly involved in? In the previous section, we described the functions possible in GreenTrells to support different graph tasks. In this section, we present procedures to complete a variety of tasks. We will pose the task taxonomy questions which we select from previous works. Understandably, there are numerous tasks defined in taxonomy papers. We focus on select tasks, as follows: 1) Identify outliers, 2) Identify duplicate nodes, and 3) Match a subgraph.

Identify Outliers

Outlier detection in graphs involves identifying anomalies with respect to a given relationship or expectation . Every node could have an outlier behavior in some circumstance. For example, an outlier node given the degree distribution versus the clustering coefficient distribution may be different nodes.

When a graph has too many nodes where using a node-link view makes little sense, one way to understand the graph is to identify outliers. The following is the GreenTrellis procedure to identify an outlier with respect to a given relationship or expectation: First: The

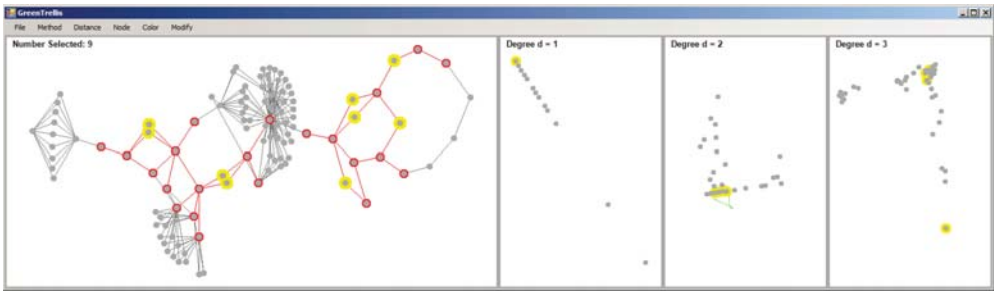

Figure 4.5: At distance 2, a group of nearby nodes are selected. They show similar features.

analyst must decide which graph characteristic for which to find an outlier. In GreenTrellis, selecting a characteristic corresponds with opening a new graph signature scatterplot.

Second: In the chosen scatterplot, the nodes farthest apart from the others represent an outlier. The analyst can visually inspect these outlier nodes. Figure **??** shows an example outlier selected from the scatterplot.

Further examination of the outliers includes comparing their signature vector annotations with the average of the rest of the nodes. Also, the analyst can examine the outlier nodes behaviors at increasing or decreasing distances. Outlier detection is a low-level visual inspection task in GreenTrellis.

Identify Duplicate Nodes

Similar to identifying outliers, we can use GreenTrellis to identify duplicate nodes. Two or more nodes may represent the same entity but due to data error, these nodes may be duplicated in the graph. If we identify duplicates, we can correct a graph. The following is the GreenTrellis procedure to identify whether a set of nodes are duplicated, essentially this involves checking if their surrounding structures are similar:

First: We can zoom into the node-link panel and visually compare their nearby structures by making selections and examining the highlighted structures.

Second: For further analysis, for example, to determine if these nodes are more similar

to each other with respect to the rest of the graph, we can utilize the scatterplots. We select a scatterplot and compare the distance between the nodes in question versus the distance between the other nodes. For example, Figure 4.5 shows several nodes with similar features in the force directed view. These nodes were identified by selecting nearby nodes in the distance 2 plot.

The analyst can utilize her discretion to determine if the visualization convinces her that the nodes are indeed duplicates. To identify if duplicate nodes exist in the graph, we use the same procedure as above, but repeated for every node in the graph. Starting at step 2 is a feasible way to solve this task for large graphs.

Visual Subgraph Matching

The subgraph matching problem entails searching for a graphs existence in another graph. This search involves graph isomorphism , which checks for matching nodes and edges between two graphs. Subgraph matching is computationally intensive but required in querying biological pathways or finding a hidden criminal ring in a social network . We present a couple ways to match subgraphs in GreenTrellis:

Method A: An analyst can search for the subgraph in the node-link layout by visually tracking the matching nodes and edges.

Method B: An analyst can sketch the subgraph and use the scatterplots to find nodes with overlapping characteristics (essentially, this involves the duplicate identification task). However, we can select a group of nodes to speed visual mapping. The signature scatterplots act as visual indexes. They digest the raw network into clustered similar structures to provide filters for faster comparison between graphs. This process is analogous to indexes used in relational databases.

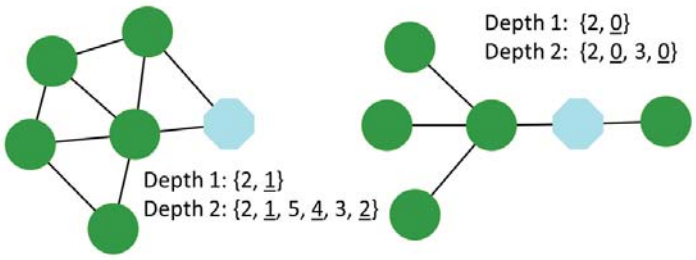

Figure 4.6: Neighborhood Signature on the Blue Nodes.

4.4 Graph Signatures

In the previous section, we presented GreenTrellis and reviewed the topological BFS signature. In this section, we take a closer look at forming new graph signatures to gain different insights into the graph, based on factors not apparent from the force-directed or the topological BFS signature plots.

4.4.1 Neighborhood Signature

Section 2 shows several examples where the topological BFS signature is an intuitive and first approach to study a graph. In this subsection, we show a new signature, called Neighborhood, which will expand our analysis capability in GreenTrellis.

We can extend the nodes local topology captured by the topological BFS signature by including the connectivity of the nodes at each level of the BFS. Including this connectivity information will remove cases, for example Figure 4.6, where nodes have the same graph signature but their neighborhood structures look different. Both nodes have a topological BFS signature of $\{2, 3\}$. For a further understanding of the similarity between nodes in the graph, we summarize the nodes neighborhood connectivity in the signature.

The subgraph matching problem is searching for a graphs existence in a larger collection of graphs. This problem may seem irrelevant to our concerns about adding connectiv-

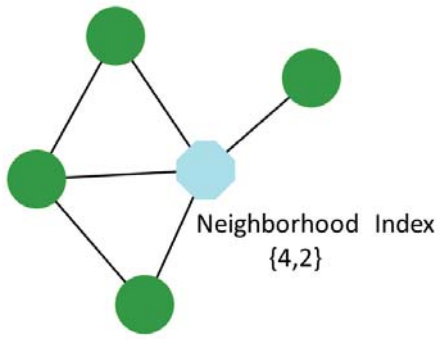

Figure 4.7: Neighborhood Index on the blue node.

ity to graph signatures to improve differentiation between node topologies. However, one method to solve subgraph matching presents an interesting structure for our use. When the collection of graphs to search against is large, filtering these nodes can accelerate the matching process. A pre-processing step in filtering nodes is to characterize them in order to filter unlikely matches. In [23], a structure is defined that stores the nodes degree and connectivity between its neighbors, as shown in Figure 4.7. The neighborhood index for the blue node is $\{4, 2\}$, accounting for 4 neighbors and 2 edges between the neighbors.

The benefit of encapsulating this structure into a graph signature is that nodes clustered nearby in the scatterplot will have more in common. Not only do they show the same topological BFS structure, but they also share the same number of edges between their neighboring nodes. In effect, we will visualize the different frequencies and clusters of similar subgraphs occurring in a graph.

Constructing the Neighborhood Signature

To construct a Neighborhood Signature, we concatenate the neighborhood structure for the nodes as we increase the BFS depth. We concatenate the new nodes reached at the current BFS level in order of decreasing degree. Next we pad the node signature with 0s to account for varying degrees in the graph. In GreenTrellis, we threshold the signature

size by concatenating only the top three largest degree nodes at each BFS level. Thus, the signature vectors will be the same size for all the nodes in the graph.

In Figure 4.6, the Neighborhood Signature at depth 1 for the left and right blue nodes are $\{2, \underline{1}\}$ and $\{2, \underline{0}\}$ respectively. We underline the neighbor edge connectivity value for visibility. At depth 2, the signature for the left and right blue nodes are $\{2, \underline{1}, 5, \underline{4}, 3, \underline{2}\}$ and $\{2, \underline{0}, 3, \underline{0}\}$, respectively.

4.4.2 Cyclic Signature

The topological BFS and neighborhood signatures only consider information gathered within the BFS levels. We did not have to survey the entire graph to construct a nodes signature; only the distance of interest is included in the signature. In this subsection, we present a graph signature with cyclic and clustering properties that consider multiple levels simultaneously. Sometimes a nodes relationship to the entire graph reveals unique characteristics. For example, any path-related feature (i.e., longest path, average path) can only be known by examining the whole graph. We introduce a graph signature that exposes a nodes cyclic properties. In other words, to answer the question of what size of a cycle is the node involved in, the computation requires that we look beyond the BFS levels because a nodes cycle may extend through the whole graph.

Cyclic properties relate to circulation and information flow in a graph. In a graph with a tree structure, the nodes will have zero cyclic patterns. We can search for nodes that exhibit large or small circulation patterns with the cyclic graph signature. Below, we show how to construct this signature:

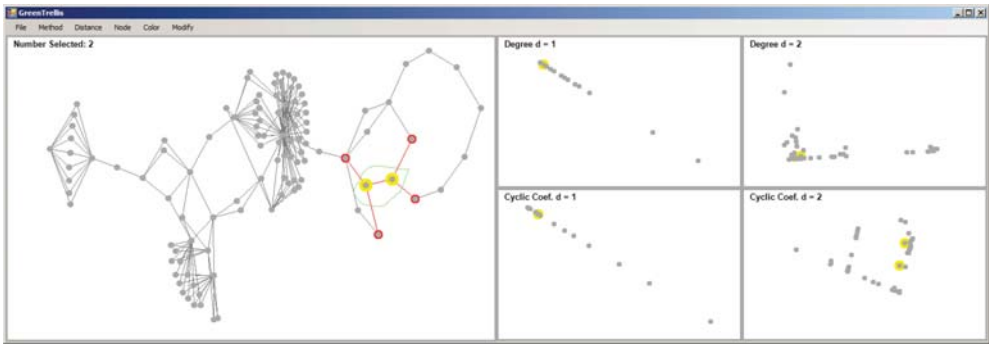

Figure 4.8: The two nodes selected in the force directed view appear to be similar in all plots except for the cyclic at distance 2.

Constructing the Cyclic Signature

We construct the cyclic signature with the cyclic coefficient metric defined in [11]. We compute the cyclic coefficient $L(n)$ for each node:

$$L(n) = \frac{2}{degree(n) * (degree(n) - 1)} \sum_{\text{l,m}} \frac{1}{S_{l,m,n}} \tag{4.1}$$

where l and m are n's neighbors and S is the smallest loop between them. The cyclic coefficient represents the average of the inverse size of the smallest loop that connects node n and its two neighboring nodes. After computing the cyclic coefficient for each node in the graph, we average the cyclic coefficients for the new nodes reached at the current BFS level. So the signature vector will contain the following values $L(n,0), L(n,1), \cdots, L(n,r)$ where

$$L(n,r) = \frac{\sum_{i \in V_R} L(i,0)}{|V_R|} \tag{4.2}$$

V_R is the subgraph induced by r hops away from node n and $|V_R|$ is the number of nodes in the subgraph.

Example Cyclic Analysis

We give another example of a graph observation made with GreenTrellis. In Figure 4.8, we select the two nodes on the force directed view. They participate in different closed loops. If we look at these points in the signature plots, they are indistinguishable in most plots, except for the cyclic signature plot. Only the cyclic signature is able to identify their different size of loops.

4.4.3 Designing Other Signatures

As the above graph signatures show, there are many ways to visualize a graph through signatures. Any graph metric can be exploited and visualized through signature plots in GreenTrellis. To do so, the metric must satisfy a locality property. To construct the neighborhood and cyclic signatures, we defined their signature values at increasing BFS levels. The increasing depth of a signature shows how the nodes local topology changes. In this subsection, we present two sample templates to encapsulate a graph metric into a signature: Aggregation and Independent Levels method.

Aggregation creates a graph signature by aggregating the nodes graph metric values of the nodes reached at the current BFS level. We can aggregate values by averaging them or taking their maximum or minimum. This method will accumulate the metric values to compute the signature value at the current BFS level. In GreenTrellis, the graph signature plots for cyclic is an example of the aggregation method.

The other method to create a graph signature is Independent Levels. The Independent Levels method computes the signature value at a BFS level based only on the current nodes at that level. The topological BFS and neighborhood signatures are examples of this method.

Neighbor Share and Clustering Signature Example

In GreenTrellis, we provide two more signatures, Neighbor Share and Clustering signature, which usually help characterize biological networks [18]. We base each signature on a metric: neighbor share and the clustering coefficient [17]. Neighbor share measures the tendency of nodes to share neighbors with other nodes. The clustering coefficient reflects the number of edges existing between a nodes neighbors. In genetic networks, nodes in normal networks usually have higher clustering coefficients than nodes in tumor networks [18]. We construct both signatures with the aggregated averaging method, similar to the cyclic signature construction.

4.5 Usability Study

In the previous sections, we described each graph signature and its use in GreenTrellis. GreenTrellis strength is to view multiple signatures simultaneously. In this section, we present a usability study to test GreenTrellis in a variety of tasks according to different test measures. In a realistic setting, the analysts are aware of the context of the graph they wish to study. They probably are not experts in graph theory or how multiple graph metrics relate to the context of their problem. We planned the user study to simulate this background as closely as possible. However, for a controlled study, we made simplifications, as follows: we described the data set and the problem-at-hand. The users task is to complete the problem with the given tools. In total, 16 different users participated in the study, including 6 summer interns at Pacific Northwest National Laboratory and 10 students from the University of Michigan. All users have a math or computer science background without graph theory training. We conduct the user study individually for each participant in a quiet room with GreenTrellis installed on a PC.

Before conducting the user study, we train each user for 10 minutes on how to under-

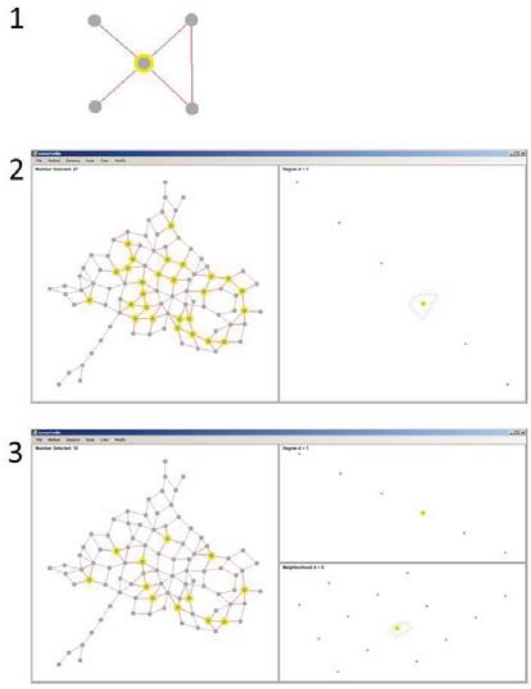

Figure 4.9: The first figure shows the subgraph to identify. The second figure shows the nodes highlighted when the analyst selects nodes with degree 4. Much fewer nodes are highlighted when the neighborhood signature plot is used.

stand and use the following features: (1) navigating the node-link panel with zoom/pan/selections/colori annotations, (2) how a scatterplot is created, (3) panels display linked graph information through simultaneous highlighting, (4) how to open a new scatterplot at any distance, (5) description of the degree graph signature and the neighborhood graph signature, and (6) how to identify an outlier in a scatterplot.

In the following subsections, we describe each task, the data sets the task covers, and the resulting test measurements. We conclude this section with a short discussion and notes on unexpected findings from the studies.

4.5.1 Task 1: Comparison between Node-Link, Degree, and More Complex Scatterplot Panels

The first task studies the following hypothesis: Using the node-link panel with the degree signature and neighborhood signature scatterplots allows an analyst to identify and

track substructures occurring in a large graph (¿100 nodes) faster and more correctly than using a node-link view alone. In this study, we lock the distance the signatures represent at one edge away from the node. We vary the panels available to the user from the following: (1) only node-link, (2) node-link with the degree method, (3) node-link with degree and neighborhood methods. We utilize a dataset which represents a freeway network, where nodes are cities and edges are freeways. There are 122 nodes. The graph is planar and has no crossing edges in the node-link panel. The user is asked to count the number of occurrences of a small subgraph (5 nodes), shown in Figure 4.9.1, within the freeway network given the three different tool scenarios, discussed above. In this study we measure the time to complete the task and the result accuracy. The following is the procedure used:

1. We present the graph description and tell the user we will show them a small subgraph and the task is to identify the number of its occurrences using only the panels provided.

2. We show the user the subgraph and the freeway network on GreenTrellis node-link panel view. We ask the user to complete the task in less than five minutes.

3. We open the degree signature scatterplot at distance one. We ask the user to complete the task in less than five minutes.

4. We open the neighborhood signature scatterplot at distance one. We ask the user to complete the task in less than five minutes.

After each step, we record the time and ask the user for the result. After the completion of the procedure, we give the user a short survey to fill.

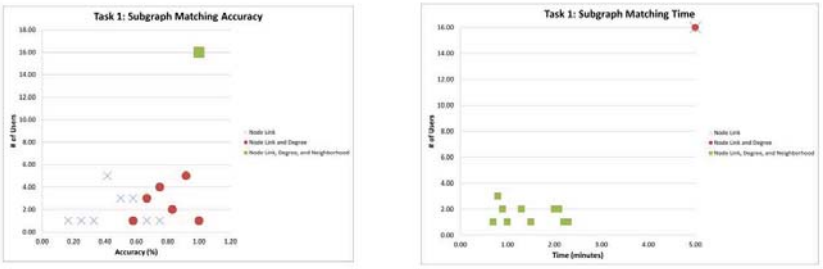

Figure 4.10: Using more signature plots results in a higher subgraph matching accuracy and faster exploration.

Results

When participants were presented only the classic node-link view, they tried to find the subgraph by finding nodes with degree four, then checking the neighbors of those nodes. When they found a match, they colored the node to keep track of a finding. One participant colored nodes which did not match as well, but only did so for six nodes. All participants utilizing only the node-link view complained that it is difficult to keep track of the matches. Nobody was able to complete the task in less than five minutes. The accuracy of results ranges from 17 to 75 percent of the subgraphs found.

When the degree scatterplot was also included, the users immediately searched in the degree scatterplot for the degree four nodes. When they found this group, they colored it. Next they searched among the colored nodes in the node-link view for the subgraph. Again, they colored the matches. Some participants were able to find all the subgraphs but not in less than five minutes. Lastly, when participants were allowed to use the neighborhood scatterplot, everyone found all of the subgraphs. Most participants first colored the nodes with degree four by using the degree scatterplot. Then among these nodes, they searched in the neighborhood scatterplot for one connection between the nodes neighbors. This process immediately gave them the answer.

Some participants struggled to see the connection between the degree and the neighbor-

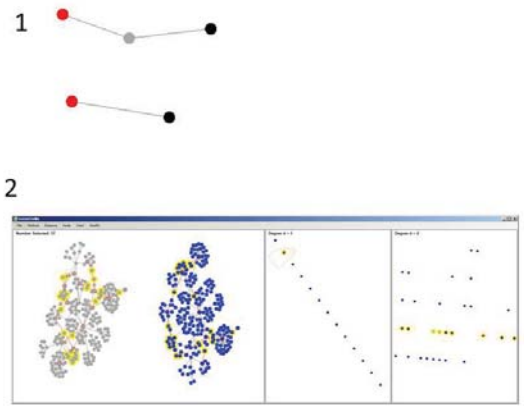

Figure 4.11: The first figure shows the modification made to the original graph. The second figure shows the graph comparison using the signature plots at increasing distances.

hood scatterplot. For example, the degree four nodes in the degree scatterplot correspond to degree four nodes in the neighborhood scatterplot. But by highlighting and checking the annotation between the degree and neighborhood scatterplots, they realized the link between the panels.

After the neighborhood scatterplot showed the result, participants noted they were not sure whether to trust the result immediately. They browsed through some of the highlighted nodes to manually compare it with the subgraph. Figure 4.9 shows the difference between using the degree plot and the neighborhood plot to identify the subgraph.

All three panels provided useful information. The node-link panel allowed the users to see the structure of each individual node and the scatterplots summarized their information. Furthermore, these results show that the complex signature scatterplot helps a user solve a task which seems impractical within a limited time with only a node-link panel. Figure 4.10 shows that using more signature plots results in a higher subgraph matching accuracy and faster exploration.

4.5.2 Task 2: The Effect of Locality on Solving a Graph Task

The previous study examined the benefits of utilizing more complex signatures to solve subgraph matching. We study the benefits of locality to complete a graph task. We lock the signature to the Degree Signature Scatterplot and vary the distance the scatterplot represents. Our hypothesis is that an analyst using a scatterplot which represents a greater distance of the nodes environment can compare the differences between two large graphs (¿100 nodes) with greater accuracy. A difference is defined as the traditional graph set difference operation .

In this task we use phone call data sets. We present the user with two different graphs. The difference between the graphs is that a node and its two edges are removed, and then an edge is added between the removed nodes neighbors, as shown in Figure 4.11.

In this task we show the user the two graphs and ask if they can identify a difference between the graphs. If they answer yes, we ask them to show us the difference. We ask these questions given different panels. We measure the time to complete the task and the correctness of the result. The following is the task procedure:

1. We describe the phone call data sets and that the two graphs may have a difference between them. We ask them to identify if there is a difference, if so, to show us the difference.

2. We show the user the two graphs on the node-link panel and ask for their findings. We allow five minutes to complete the task.

3. We include the distance 1 degree scatterplot and ask for their findings. We allow five minutes to complete the task.

4. We include the distance 2 degree scatterplot and ask for their findings. We allow five minutes to complete the task.

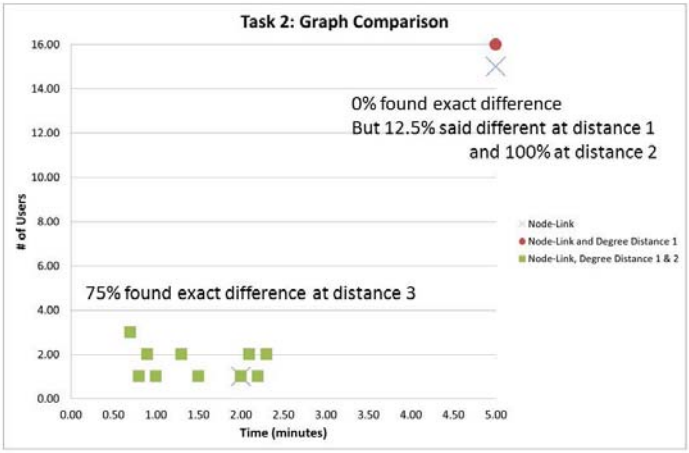

Figure 4.12: Using a greater distance helps identify the differences between the graphs.

After each step we record the time and the results. After the study we ask the user to complete a survey.

Results

Participants found it difficult to complete the task with only the node-link view. Some participants looked for general topological differences but could not identify any differences. One participant highlighted the subgraphs individually and found that there is a difference because one graph has an extra node (shown by the number of selected nodes). On the other hand, another participant noted, "I don't even know where to start." However, when we included the degree distance 1 scatterplot, all participants found the graphs are different.

The general process participants used to find that the graphs are different is they highlighted each group in the scatterplot. When they found a group with an odd number of nodes selected, they concluded the graphs are different. No participant was able to find the exact node difference until the distance 2 scatterplot was presented.

When the distance 2 plot was shown the participants first tried to understand the link be-

tween the distance 1 and distance 2 panels. They transferred the information they learned from the distance 1 panel by highlighting the nodes that convinced them there is a difference between the graphs. Participants then zoomed into the distance 2 view in the area these nodes are highlighted. They saw a color difference. When they selected the nodes with a color difference they determined the exact node difference between the two graphs.

The results from this task show that participants were able to find the difference between two large graphs by using information available from the distance 2 scatterplot. By including greater distances which the signature represents, they were able to complete the task. The results are summarized in Figure 4.12.

4.5.3 Task 3: Selecting the Appropriate Signature for Graph Characterization

The two previous studies tested GreenTrellis' utility in different graph tasks. We controlled the distance and the scatterplots for these studies. The final study examines how effective GreenTrellis' functionality is for an analyst to determine an appropriate scatterplot given the problem. Therefore, we allow the analyst to determine which locality and method to use. Our hypothesis is: an analyst is able to determine an appropriate signature to use to study their problem in GreenTrellis by examining the scatterplots.

We reuse the freeway network for this task. We tell the user that they will read a description and the task will be to identify the appropriate method to further examine the problem. We measure the time to complete the task and whether the user chose the correct graph signature. The following is the problem description, "You are an analyst studying a freeway network. A vehicle safety company would like to know how to distribute safety booths across the cities. Before they do, they would like to understand which cities are harder to reach. For example a city on the exterior with only one freeway is a difficult to reach city, and a city with many freeways to other cities which also have many freeways is easier to reach. Which scatterplot do you think reveals this information about the cities

the best?"

In this case, the answer is the neighbor share method. The users do not have any training or previous experience in using this method. However, they are aware of how to select a new method and utilize the tool by this point. The reason neighbor share is the correct method is because it is a topological coefficient which represents the likelihood a nodes neighbors are also shared by other nodes. Nodes with low neighbor share values represent cities which are harder to reach because their neighbors are not popular either. Neighbor share is a coefficient which compares node reachability on a global scale.

In summary, the procedure for this task is the following: (1) the user is asked to read the description, (2) the user is given time to complete the task. After the study we give the user a short survey.

Results

All participants utilized the maximum five minutes to complete the task. All but one participant found the correct signature, which is neighbor share. The participant who chose the wrong signature selected cyclic.

We observed that participants selected nodes which they expect a certain behavior. For example, they selected the node in the node-link panel which should have an outlier low neighbor share value because of its environment. Also, they selected interior nodes with many neighbors, expecting a high reachability value. They compared these selections to the highlighted nodes in the signature scatterplots. These small selections acted as a ground-truth for them to understand how the signature scatterplot relates to the context of their problem. Once they saw neighbor share best captured the ground-truth, they trusted the methods characterization of the other nodes in the graph.

Participants also read the descriptions of the methods. However, we were careful to use different wording in the problem description so that they would be required to use the

tool. In this task, we allowed the user to choose from five ways to characterize the graph structure given the context of the problem. We do not argue that this list represents all of the possible ways to represent graph structures. The purpose of this study is to show that a novice user who does not have graph topological theory expertise can utilize the tool to find an appropriate signature to study the graph.

The results of the study show that the user can discover the appropriate signature by reading node annotations and interacting with the graph in GreenTrellis.

4.6 Contributions

We presented a graph exploration tool, GreenTrellis, which provides locality-driven multi-modal graph signature analysis. We also designed several new rich signatures to capture different topological characteristics of a node. We demonstrated with examples and user studies over several datasets the different uses of the tool. Our presentation was focused around making a tool which can help analysts understand the graph structure, compare the graph features at any distance away from nodes, and a tool with customizable features (drawing a graph, selecting localities, signatures to display).

GreenTrellis strength lies in concurrent highlighting, which reveals node similarities, differences, and other graph features. A benefit of this type of analysis in GreenTrellis is that it does not give the user a cryptic graph metric on a node to define its characteristics. Instead, we visually comprehend their similarity by zooming in and out of their signature scatterplots.

The scatterplots are ordered in increasing signature depths. GreenTrellis supports discovery exploration for different classes of questions: known-known, known-unknown, unknown-known, and unknown-unknown. These cases range from knowing the existence of a set of nodes and their common characteristics to not knowing the set of nodes nor

their common characteristics. For example, in a known-known case we already have an identified set of nodes and a set of shared characteristics between these nodes. In an unknown-known case we search for a set of nodes similar to our already identified set of nodes for similar characteristics (i.e. we are unsure if there exist such a set of nodes). To search for these new nodes, we select a group of nodes in the graph and then examine the signature plots in GreenTrellis to find common characteristics between these nodes. The unknown-unknown case deals with the reality that the users have to browse the entire graph (both usual and unusual structures). For example, we know that a scenario may exist between the nodes, but we do not know which set of nodes participate or their common characteristics. Our tool facilitates the brute-force browsing approach by categorizing the data based on different characteristics. Users can sample individual items of the clustering and learn the characteristic of the group or population (based on the scatterplot matrix). Having an array of matrix is necessary to cross-interrogate the graph because only one scatterplot may not reveal a unique trend.

4.7 Future Work

Currently GreenTrellis focuses on computed attributes. We visualize outliers and communities surrounding topological features. Topological features enable us to solve difficult graph tasks such as subgraph matching, duplicate node searching, and other problems we examine in the examples and user studies in this chapter. However, there are multitudes of intrinsic attributes which describe characteristics in the context of the dataset (e.g. height, color, etc.) For future work, we would like to create intrinsic attribute signatures which are also locality meaningful. This step involves identifying ways to quantify discrete attributes and understand how to represent frequencies of various intrinsic attributes at growing distances away from nodes. This future work will allow analysts to study intrinsic and

computed attributes in signature scatterplots.

CHAPTER V

Uncertain Data

5.1 Introduction

In many applications only attributed graphs can capture the complicated data structure naturally and intuitively, with nodes corresponding to the entities of interest, and edges to relationships and high dimensional information on both. Due to the increasing demand of management and analysis of graph data, various works have proposed new graph database systems. However, more difficulty arises in describing and using graphs if there exists some uncertainty within the analytic process.

As far as we know, one less explored but equally important problem is the uncertain query in graph analysis. Outside of networks, an example of an uncertain query is when the exact spelling of a word is uncertain in a web search. This problem is well studied and its solution is utilized in many search engines. In a network setting, this problem occurs, for example, when an analyst is unsure of the connectivity or the attribute values on the nodes and edges of a sub-network she would like to find in a network. There are several motivating applications for this type of uncertainty:

Usually on a heterogeneous molecular network with proteins and genes there exists uncertainty over the name or the number of interacting partners of a molecule. A scientist may have some idea about what values and structure she expects. But to manually track

her uncertainty from the conception of the uncertain subnetwork query to the multiple results is a tiring task. This management problem is even more confounding when there are multiple stages in network analysis. At any point in the workflow, there may have been an uncertain query. The analyst has to manually determine how likely a resulting graph represents the queries she made over the whole workflow.

Another motivating application occurs in social network intelligence, for example in [46], which studies a criminal ring in a phone call network, we are given uncertain information: the subnetwork may consist of five individuals. One of the individuals $might$ be identifier 200, who may speak with another individual very frequently. The possible hypotheses social sub-networks are described with uncertainty on degree, connection, and attribute values. The challenge was to identify the most likely social network based on the graphs returned by the different hypotheses social networks.

5.2 Introduction

In this chapter, we provide a model for the uncertain predicate in a network query setting. This model guides the user to define the uncertain query. Then it generates results, with ranking based on the user's preferences. We build the uncertainty querying over a visual analytic algebra. We show how the uncertainty on the queries can be composed over an analysis workflow.

Our uncertainty model follows this basic outline: First a user inputs an uncertain query. Next the system expands this query into a set of exact queries representing the uncertain query. Then the user places a probability on each exact query which notes how well the exact query captures the uncertain query. The system then computes the graph manipulation operation and probabilities for the results.

In the uncertain query problem we assume the graph itself is exact. The uncertainty

remains only at the query level. The rank value associated with the result of an operator is stored as a computed graph attribute and denotes how well the outcome describes the user's uncertain query. However, this value does not introduce uncertainty on the graph elements and attributes' existence. This distinction allows us to solve the uncertain query problem without delving into uncertainty on the graph.

The following is an outline of the chapter: In Section 5.3 we present related work. In Section 5.4 we state the problem. In Section 5.5 we describe our uncertainty method. Section 5.6 describes how uncertainty can be used in a workflow. We conclude in Section 5.8.

5.3 Related Work

There is a large body of research on uncertain databases and graphs due to the growing demand for applications that require management of uncertainty within data mining or the information retrieval processes. The uncertainty problem can be found in both database and the query space. Trio [13] is database system that supports uncertainty and lineage within the data.

In order to solve the uncertainty on the graph data, Getoor et al.[24] extend the probabilistic relational model by modeling interactions between attributes and link structures. This approach combines the attribute generative model with the relational generative model to improve attribute predictions. In Getoor's survey paper [23], she further provides a complete overview of popular link mining methods to construct uncertain networks. Unlike mining and prediction on uncertain data, our method only assumes the uncertainty with the query space. In order to attack uncertain data problem under a different motivation, Sarawagi et al.[26] extract structured entities from unstructured sources by attaching a probability score indicating the accuracy of the extracted entities. Even though the proba-

bility score is constructed to model the uncertainty within the attributes of dataset, the idea is similar to placing the probability score on the results from an uncertain query.

The uncertain query problem is often solved using query relaxation or query extraction. Google Suggest, a service powered by Google for phrase suggestion, and YourEye [7], a real-time phrase recommender, are systems that suggest related phrases to the incomplete user query. YourEye selects the related phrases based on previous queries. The advantage of YourEye system compared to Google Suggest is supporting prefix phrase suggestions.

Driven by web search engine applications, Zhang et al.[64] provide a fast query recommendation algorithm by computing query cluster. The query cluster can be constructed based on users sequential search behavior or content based similarity within the given query. Barouni-Ebrahimi et al.[6] expand the query recommendation method further by ranking outputs based on user click-through feedbacks as well as frequent phrases from the past queries. Michelson et al.[41] propose his query expansion method for a search problem based on mining a set of heterogeneous transformations. Similar to other query recommendation methods for information retrieval, we utilize content similarity to transform the uncertain predicate into a set of exact predicates. We do not need to consider information from previous queries while applying relaxation thanks to the independent assumption.

Besides applying query relaxation methods to information retrieval, flexible query answering on graph modeled data [39] is a query method for the case when a complete knowledge in data sets is lacking. It gives an approximation on the predicate values first and then predicting the graph structure. But this method relies on an expert to define meaningful relationships between nodes and several scoring functions. There has been other work that provides similar structures to a query, especially if it is difficult to find the exact match for an exact predicate. Yan et al. [63] attack the similarity search of complex

structure problem by using a relaxation ratio of a query graph. Our work relies on similar methods to generate exact queries given an uncertain query, but besides considering only single query search step, our method is able to rank outputs based on multiple query searches.

5.4 Background And Problem Statement

In this section, we formally define the uncertainty problem we address. Our uncertainty model is based upon the graph visual algebra introduced in [47]. We briefly review the operators and model in the graph visual algebra which are relevant to this chapter. Next, we present the problem statement.

5.4.1 Graph Model And Assumptions

The central object of interest is a graph set, defined as $D = \{D^1, \ldots, D^n\}$ and $D^i = [G^i, X^i]$, where $G^i = (V^i, E^i)$ is a standard graph with vertex set V^i, edge set E^i, and attributes set X^i. The attribute structure X^i associated with graph G^i has three components: $X^i = [X^i_V,\ X^i_E,\ X^i_G]$, where X^i_V contains node attributes (e.g. in- and out-degree), X^i_E edge attributes (e.g. edge betweeness, direction) and X^i_G graph attributes (e.g. diameter).If there is only one graph in the graph set G, index i may be left out during the analysis. The attributes can be either intrinsic or computed type. In order to obtain computed attributes there exists a collection of user defined composition functions $\mathcal{F}$.

5.4.2 Graph Algebra

Predicate

The predicate language is used to define the queries a user would like to manipulate. A predicate tuple is $\rho = (V, E, \mathbb{X}_V, \mathbb{X}_E, \mathbb{X}_G, !E)$. The $\mathbb{X}_V, \mathbb{X}_E$, and $\mathbb{X}_G$ in the predicate are condition lists on the attribute values on nodes, edges and graphs. In the tuple $!E$ is the excluded edge list. An $exclusion$ on edge e specifies that e must not exist in the graph G.

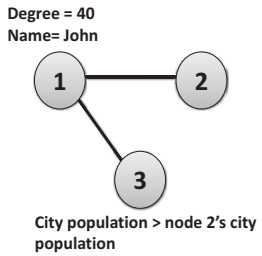

Figure 5.1: Simple Predicate Graph ρ_1

Example V.1. To express the predicate shown in Figure 5.1 where a node with $name = John$ and $degree = 40$ has two neighbors. The neighbors' city population is described such that one lives in a higher populated city than the other. The predicate tuple is $\rho_{5.1} = (V = (v1, v2, v3),\ E = (e_{1,2}, e_{1,3}),\ \mathbb{X}_V = ((X_{v1.degree} = 40),\ (X_{v1.name} = ''John''),\ (X_{v3.city\_size} > X_{v2.city\_size})),\ \mathbb{X}_E,\ \mathbb{X}_G,\ !E)$.

A complex predicate can be constructed by using logical functions, such as AND and OR function. For example, if user wants to merge people or nodes by gender, one can use a complex predicate ρ=$\{\rho_1 = v_1, X_{v1.gender} = Female$ AND $\rho_1 = v_1, X_{v1.gender} = Male\}$.

Witness

A witness is an attributed graph that satisfies the conditions and structure of a predicate. Both structural and attribute properties of D, and the predicate, ρ, must match for D to be a witness of ρ. There must be a bijective mapping between the vertices and edges in the predicate structure and the witness.

Figure 5.2 shows an example witness for predicate ρ_1. This witness satisfies the structural and attribute conditions of ρ_1, shown in Figure 5.1.

Graph Matching Function

The graph matching function, γ, returns the witnesses in a graph given a predicate. It is a function used by the other operators when it is necessary to find the witnesses in the

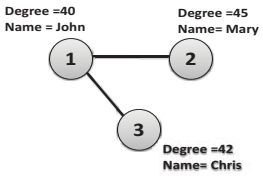

Figure 5.2: Witness to predicate ρ_1.

attribute graph.

Visualization Operator

The visualization operator, $\mathcal{V}$, provides a visual representation of the target The visualization function can take various forms, including different ways of laying-out graphs, e.g. force-directed, hierarchical, hyperbolic, and also presenting the attribute data (e.g. histograms for numerical attributes, bar-graphs for categorical ones, etc...). In addition, it is assumed that the visual functions can be composed and thus produce multiple and possibly linked displays. However, since the visual function does not provide user the ability to manipulate and change the data structure. It is similar to a reset function which provides users a fresh overview of the data at the time.

Data Manipulation Operators

The algebra has selection operator and aggregation operator, which operate on sets of attributed graphs and output sets of attributed graphs.

Selection There are two selection methods which operate on sets of graphs. One method works at the element level (i.e. affecting the nodes and edges within the graph) and the other at the graph level (i.e. affecting the entire graph). Both methods take a single predicate as input.

Uncertain predicate	$\alpha*$
Exact predicate for uncertain predicate $\alpha*$	α_i
Probability set w.r.t. an uncertain predicate $\alpha*$	$\Omega_{\alpha*}$
Witness list for an exact predicate α_i	W_{α_i}
Output set based on uncertain predicate $i*$	O_{i*}

Aggregation Element aggregation merges the nodes and edges of an attributed graph by groups. The groups are described by a set of predicates. Given a set of attributed graphs and a set of predicates, element aggregation modifies the input graphs by creating a new node for each witness found for each predicate. The new node represents a merged super-node. The attributes of the new node are aggregated values of the nodes contained in the witness. The super-node also maintains the connections to other nodes in the graph. After the element aggregation is performed for each predicate group, the nodes that are now represented by super-nodes are removed.

5.4.3 Problem Statement

The problem we consider is visual analytics on uncertain queries. To be able to specify these manipulations we use the visual algebra operators. We extend the algebra to include uncertainty on predicates, in terms of uncertain values for attributes and edge existence. Our problem is to support this situation by providing the ability to evaluate uncertain predicates in a systematic and intuitive manner.

Table 1. shows the notation we will use throughout the chapter.

5.5 Uncertainty Method

5.5.1 A Visual Algebra with Uncertain Predicates

In this section, we extend the visual algebra to permit uncertain predicates for a given graph set. The basic idea is to consider an uncertain predicate as the union of multiple certain predicates, but with a probability associated with each certain predicate. We show how these probabilities are correctly propagated through operator evaluation while keeping

the closure property of the existing algebra.

5.5.2 Define Uncertain Predicate Language

An exact predicate tuple is $\rho = (V, E, \mathbb{X}_V, \mathbb{X}_E, \mathbb{X}_G, !E)$. We extend the predicate language to specify attributes and edges which contain uncertainty.

For attribute uncertainty, the condition lists are modified to include an uncertainty. The condition lists, $\mathbb{X}_V$, $\mathbb{X}_E$, and $\mathbb{X}_G$, in the predicate can have conditions now of the form "attribute_name $\approx$ value," where $\approx$ denotes that there exists uncertainty on this attribute. Similarly, edge existence uncertainty is denoted with an $\approx$ prefix over an edge in the edge list, E.

5.5.3 Process of Turning an Uncertain Query to an Exact One

Given an uncertain predicate, α^*, either the user or the system generates a set of disjoint exact predicates, $\{\alpha_1, \ldots, \alpha_j\}$ which characterizes the uncertain predicate. If the system performs this mapping, pre-defined substructure similarity measures can be used [63]. In the case of numerical attributes, Euclidean distances can be used, with a simple threshold dissimilarity.

Next the system requests the user to assign a probability to each element in this set of exact predicates. This probability corresponds to how well the exact predicate captures the uncertain predicate structure. We denote the probability set $\Omega_{\alpha^*} = \{p_{\alpha_1}, \cdots, p_{\alpha_j}\}$ where p_{α_i} is the probability for exact predicate α_i with respect to uncertain predicate α^* based on the graph at the time. Each p_i follows a bernoulli distribution with the probability of success equals to $P(\alpha_i = \alpha^*)$, therefore, all p_i follow complete separated distributions.

This procedure is illustrated below with the following examples.

Example V.2. Given an uncertain predicate $\alpha^* = \{V = v_1, X_{v_1.age} \approx 4\}$ and a friendship network G_0, a user is interested in selecting people (i.e. nodes) in the network with age

around 4. We can create a computed attribute to represent the range of the age attributes. In this example the range of the age attributes is from 2 to 6. The system returns a set of exact predicates within a distance. For example, if we set the threshold distance as a quarter of the variable range, 1, the system will return the following exact predicates:

$$\alpha_1 = \{V = v_1, X_{v_1.age} = 3\}$$

$$\alpha_2 = \{V = v_1, X_{v_1.age} = 4\}$$

$$\alpha_3 = \{V = v_1, X_{v_1.age} = 5\}$$

Based on this set, the user will define the probability set $\Omega_{\alpha*} = \{p_{\alpha_1}, p_{\alpha_2}, p_{\alpha_3}\}$ where $p_{\alpha_i} = P(\alpha_i \text{ correctly describes } \alpha^*)$ and all p_{α_i}s are independent from others.

Example V.3. Figure 5.3 shows a predicate with uncertainty on the node's color attribute and a graph G_0. A set of exact predicates can be generated by using the G_0's overall color range.

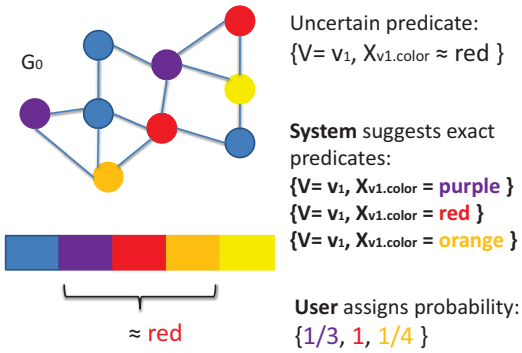

Figure 5.3: Example to transform a categorical uncertain predicate into a set of exact predicates.

Example V.4. Given an edge uncertain predicate where a user tries to select a graph structure with an $\beta^* = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, \approx e_{13}\}, x_{v1.age} = 3\}$, where $\approx$ in front of edge e_{13} represent uncertainty in the predicate structure. There are two suggested exact predicates in the set. β_1, $\{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, x_{v1.age} = 3\}$, suggests a fully connect graph,

while β_2, $\{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}\}, x_{v1.age} = 3, !e_{13}\}$, suggests only partial connected network in Figure 5.4.

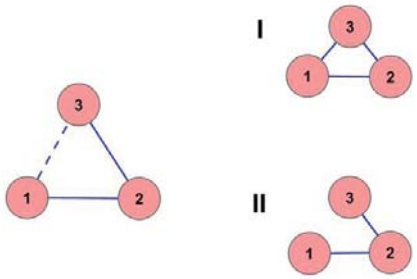

Figure 5.4: Example of edge certainty. Dash line stands for uncertain connections

Instead of attaching confidence scores to the output graphs once they are returned, we allow users to define the probabilities at an earlier stage. By defining probabilities at the predicate level before matching, the user is given more flexibility and control over the results. For example, the user can assign a probability of zero to unrelated exact predicates. This significantly improves the efficiency of the system by limiting the unnecessary graph isomorphisms. If the user does not assign any probability values to an exact predicate set, the system will assume the same probability value one to all exact predicate structure. It implies that all exact predicates in the set can correctly capture the given uncertain predicates.

Furthermore, by attaching probabilities to only a few exact predicates rather than the entire possible output set for an uncertain query, we may reduce the workload for users in assigning probability of match. Moreover, in order to further reduce the computation complexity, the system can force either the size of exact predicate or the size of output graph fall below a predefined threshold.

5.5.4 Solution Generation

After we transform an operator's uncertain predicate into a set of exact predicates with user defined probabilities, we can solve the uncertain querying problem as multiple exact graph matching problems.

For each exact predicate, α_i, we call the graph matching function γ, which returns a list of witnesses $W_{\alpha_i} = \{W^1_{\alpha_i}, \ldots, W^n_{\alpha_i}\}$. By definition, a witness is an attributed graph that satisfies the conditions and structure of the exact predicates after a single operator step, we can carry the user defined probability for the exact predicate over to its witness distribution. Whereas after multiple operator calls, we can carry the joint probability of all previous exact predicates over to their witness distribution, since the witnesses at this time should satisfy all conditions and structures of previous steps.

The output graphs depend on the operator at hand. Depending on the operator, graph, and predicates, some of the output graphs may contain several different witnesses (corresponding to different exact predicates). We compute the output graph probability based on the witnesses present by using the witness distribution.

Definition V.5. The output graph probability is defined as follows:

$$\begin{aligned} P(\text{output}) &= 1 - P(\text{no witnesses}) \\ &= 1 - P(\text{no witness in } W_{\alpha_1}) \cdots P(\text{no witness in } W_{\alpha_k}) \end{aligned}$$

where $W_{\alpha_1}, \cdots, W_{\alpha_j}$ are the witness lists found in the output.

Since all witness probabilities follow different independent distributions, we can break down the joint probability into a product of multiple user defined probabilities. The probability of this particular output can be interpreted as the probability that an output contains at least one matching for predicates α_1 to α_k. The detailed calculation will be further demonstrated in the examples below.

Example V.6. This example describes Figure 5.5 which illustrates an element selection operator given an uncertain predicate on graph G_0. The uncertain predicate is: α^*=$\{V = \{v_1, v_2, v_3\}$, $E = \{e_{12}, e_{23}, e_{13}\}$, $X_{v_1.\text{age}} \approx 4\}$. An element selection given α^* will return a triangle structure of nodes where at least one node has age attribute ≈ 4. We detail the entire element selection process given this uncertain predicate.

1. The system generates a set of exact predicates. The exact predicates are shown in Figure 5.5 part B:

$$\begin{aligned}
\alpha_1 &= \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v_1.\text{age}} = 3\} \\
\alpha_2 &= \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v_1.\text{age}} = 4\} \\
\alpha_3 &= \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v_1.\text{age}} = 5\}
\end{aligned}$$

2. The user assigns probabilities for each of the exact predicates. This step is shown in Figure 5.5 part B: $\Omega_{\alpha*} = \{\frac{1}{3}, 1, \frac{1}{2}\}$, where α^* can be described by α_1 and α_3 correctly $\frac{1}{3}$ and $\frac{1}{2}$ of the time, while α^* can be captured by the condition listed in α_2 completely.

3. Next, the element selection operator is called. Within this operator, the graph matching function γ returns three witness lists with their witness distribution. Since each of witnesses is the output of only one exact predicate, the probability of witness equals to the probability of the exact predicate in this example. This step is shown in Figure 5.5 part B.

4. Finally, the element selection operator returns the output graphs with probabilities. This step is shown in Figure 5.5 part C. The output probabilities are calculated as

follows:

$$P(O_{\alpha_1}) = 1 - (1 - \frac{1}{3}) = \frac{1}{3}$$
$$P(O_{\alpha_1\alpha_3}) = 1 - (1 - \frac{1}{3})(1 - \frac{1}{2}) = \frac{2}{3}$$
$$P(O_{\alpha_1\alpha_2\alpha_3}) = 1 - (1 - \frac{1}{3})(1 - 1)(1 - \frac{1}{2}) = 1$$

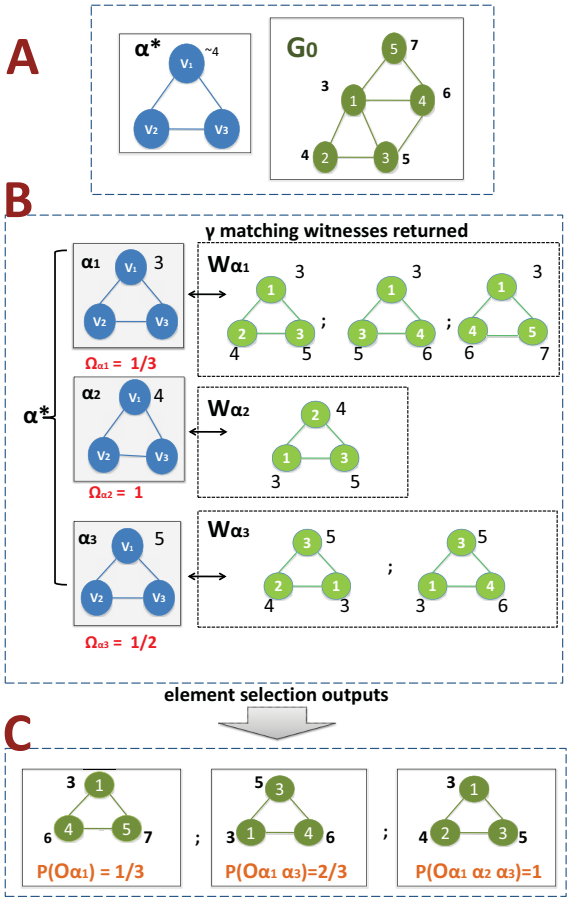

Figure 5.5: Example of element selection based on an uncertain predicate. The numbers inside the node denote the id. The numbers outside the node denote the age attribute's value. The different colors are used for visualization clarity. The dark green represents input and output graphs. The blue represents predicates. And the light green represents witnesses.

Since all the output probabilities are constructed based on witness distributions, which corresponds to the correctness of capturing the uncertain query, we are able to compare all the outputs from the same operator. In this example, the output graph from the element

selection with probability one is the best output which closely captures the user's intention. Moreover, we treat the computed probability for each output as the computed graph attribute to preserve the closure property of the algebra, since the input of our algebra framework is an attribute graph set and the output is also an attributed graph set.

5.5.5 Uncertainty Model for Complex Structural Predicates

We have introduced the uncertainty model for simple uncertain predicates. Now we describe the process for a complex structured uncertain predicate. A complex uncertain predicate is an uncertain predicate which contains more than one uncertainty and it often appears as multiple predicates with joined by logical functions such as AND and OR. We would like to simplify the complex structural predicate problem into the simple predicate problem, which we solve uncertain querying for above.

Note that there is no general solution to simplify the predicate for an aggregation operator into smaller fractional structures and still preserve the same output. This fact is because aggregation operators may change the topology of the graphs.

Example V.7. This example describes Figure 5.6 which illustrates an element aggregation operator given a complex structure predicate on graph G_0. The complex predicate list contains two predicates: α=$\{\alpha_1$ AND $\alpha_2\}$ where $\alpha_1 = \{v_1, X_{v1.color} = Red\}$ is a red colored node, $\alpha_2 = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}\}$ is three connected nodes. An element aggregation operator based on the complex predicate, α, for the given G_0 will return only one node as in Figure 5.6 output I, since we merge all red nodes and triangular structures all together into the same group ,node v_8 , defined by α_1 and α_2. However, if we simply transform the complex step into two smaller steps, where we apply α_2 first and α_1 second. The system will return different outputs as in Figure 5.6 output II, since we merge all red nodes in G_0 to create a super node, v_9, in the first step will destroy the triangular structure

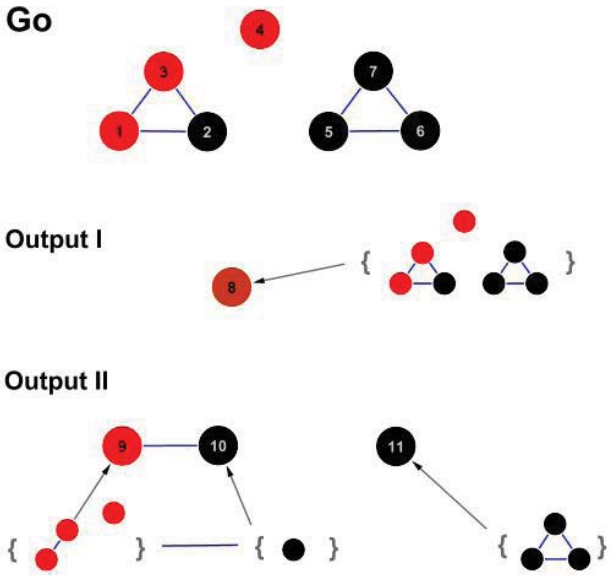

Figure 5.6: Example of element aggregation.

on the left. As in figure 5.6, all newly created super nodes are pointed by nodes

Selection operators given a complex predicate list can be broken into smaller fractions while preserving the same outputs; therefore, we may be able to simplify the complex structural uncertain selection problem under some predefined assumptions. Similar to the uncertainty model for simple predicates, probabilities of the final outputs from complex uncertain predicate are still constructed from the witness distribution. Since we are only dealing with single operators at this time, the witness distribution is equal to the probability distribution of the exact predicate set. By definition of the complex structural uncertain predicate contains multiple uncertain variables, attributes or structures. The probability distribution of the exact predicate set lives in a multidimensional space, one for each uncertain variable. In other words, the probability distribution of the exact predicate set is a high dimensional joint probability for multiple uncertain variables. Therefore, by providing additional assumptions on the high dimensional probability distribution, we can break

the high dimensional joint distribution into smaller marginal or conditional independent probability distributions. Users will be able to work only in the low dimensional space and consider the probability to describe the behavior of one uncertain variable at a time.

If there exists some degrees of dependence among uncertain variables within the complex uncertain predicate, the high dimensional probability of the complex predicate may be further broken into conditional independent components with some constraints. For example, a user tries to select people with approximately high income (I), college education level (E) and with high percentage of house ownership (O), where three uncertain variables tend to be highly correlated, the user cannot simply break the probability of the uncertain predicate into to independent components. However, instead of using only the joint probability with three variables, $P(I$ AND E AND $O)$, the system can reduce the user's workload by allowing the conditional probabilities, $P(E)$, $P(I|E)$ and $P(O|I)$. In this case, the house ownership is independent from education level given income information, but in order to keep the consistency over selection operators, the defined conditional probabilities need to satisfy the $P(I$ AND E AND $O) = P(E)P(I|E)P(O|I)$ constraint. The reduction of the user's workload can be easily explained in the follow situation. For example, for each of the uncertain variables, $\approx$ high income, $\approx$ college education level and $\approx$high percentage of house ownership, can be transformed to three exact values, such as $\approx$ college education level equals to Master degree(MA), Bachelor degree(BA) and Associate degree(AA), the user needs to define probabilities for 27 elements in the high dimensional case, but only 9 probability values in the conditional independent case under a correct constraint.

If a complex uncertain predicate can be broken into smaller independent structures, the probability space of the complicated predicate is equal to the product space of the probabilities of smaller uncertain components. This fact is supported by the independence

between each uncertain predicate designed by the user. By the commutative property of multiplication, we can reorder the composed selections and preserve the same output probabilities. Figure 5.7 illustrates how a composed element selection can be broken into two components. Since independent selection steps based on simple uncertain predicate can be reordered and still preserve the same output with consistent output probability, we can pick the optimal selection ordering to minimize the system's computational complexity.

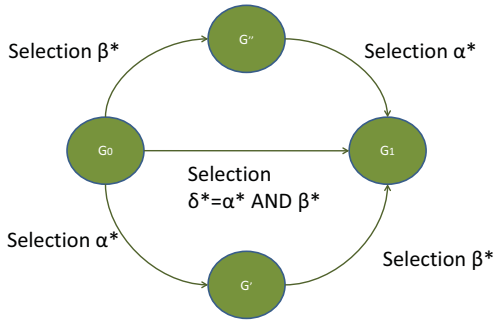

Figure 5.7: Example of joint element selection. The selection operator with uncertain predicate $\alpha*$ AND $\beta*$ is split into two. One split selection has uncertain predicate $\alpha*$ and the other has uncertain predicate $\beta*$

Example V.8. To illustrate the use of the uncertainty querying method in the independent case, we present the following example. A study has a complex uncertain predicate with a triangle node structure with uncertainty on two attributes, shown in Figure 5.9. We show an element selection operator called twice. The first time we call element selection with the predicate $\delta^* = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v1.age} \approx 4, X_{v3.age} \approx 6\}$ shown in Figure 5.9. The second time, we call two element selections in a serial fashion. The first with predicate $\alpha^* = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v1.age} \approx 4\}$; and the second with predicate $\beta^* = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v3.age} \approx 6\}$(shown in Figure 5.8). The latter, two element selection calls will produce the same outputs as the single element selection call with predicate δ^*.

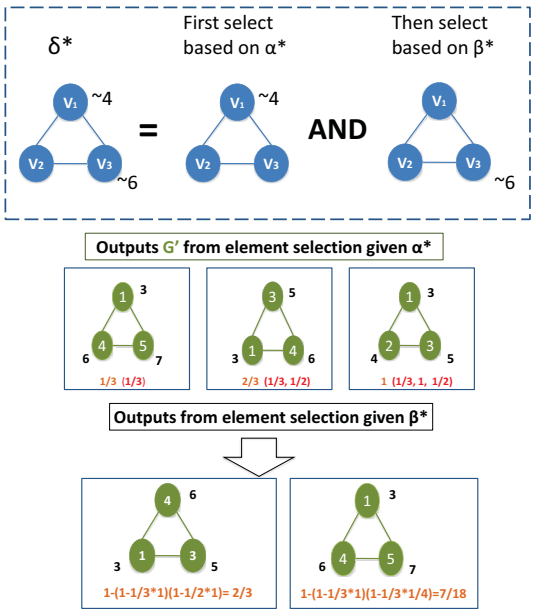

Figure 5.8: This figure shows that we get the same outputs as the combined uncertain predicate structure if we split the predicate structure into $\alpha*$ and $\beta*$ and perform two serial element selections.

Figure 5.9 part A shows an attributed graph G_0. This graph represents a social network of friends where nodes are people and edges represent a friendship between two people. The numbers inside the nodes represent an id; the numbers outside the node are the age attribute's value. Graph G_0 has the identical graph structure we use in the examples throughout the chapter.

A user would like to call an element selection with the uncertain predicate δ^* shown in Figure 5.9 part A. Using the process outlined in Section 5.4.2, the γ function generates the witnesses shown in Figure 5.9 part B. Prior to generating the witnesses, the system

generates a set of exact predicate,

$$\delta_1 = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v1.age} = 3, X_{v3.age} = 6\}$$

$$\delta_2 = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v1.age} = 4, X_{v3.age} = 6\}$$

$$\delta_3 = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v1.age} = 5, X_{v3.age} = 6\}$$

$$\delta_4 = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v1.age} = 3, X_{v3.age} = 7\}$$

$$\delta_5 = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v1.age} = 4, X_{v3.age} = 7\}$$

$$\delta_6 = \{\{v_1, v_2, v_3\}, \{e_{12}, e_{23}, e_{13}\}, X_{v1.age} = 5, X_{v3.age} = 7\}$$

and requests user to input 9 probabilities for exact predicates, $\Omega_{\delta^*} = \{\frac{1}{3}, 1, \frac{1}{2}, \frac{1}{12}, \frac{1}{4}, \frac{1}{8}\}$, corresponding to the uncertain predicate. Finally, the system outputs the graphs and probabilities shown in Figure 5.9 part C.

With this example, we now show if we decompose the complex uncertain predicate to the following simple predicates: α^* equals to $\{V = \{v_1, v_2, v_3\}, E=\{e_{12}, e_{23}, e_{13}\}, X_{v_1.\text{age}} \approx 4\}$ as in the previous example and β^* equals to $\{V = \{v_1, v_2, v_3\}, E = \{e_{12}, e_{23}, e_{13}\}, X_{v_1.\text{age}} \approx 6\}$, and perform serial element selections, we receive the same outputs as before. Figure 5.9 shows an element selection with uncertain predicate α^* called first and its outputs. Next, an element selection with uncertain predicate β^* is called. The outputs in Figure 5.9 part C and the outputs generated by β^* (after α^* is called) are the same. By applying serial element selection over one complicated selection step, we effectively cut down the user's workload from defining a probability set $\Omega_{\delta*}$ which contains 6 probability scores to construct two simple probability sets $\Omega_{\alpha*} = \{\frac{1}{3}, 1, \frac{1}{2}\}$ and $\Omega_{\alpha*} = \{1, \frac{1}{4}\}$.

In addition, a complex predicate list may contain both uncertain and exact predicates. The exact predicates can be treated as a special case of an uncertain predicate, i.e. probability of 1 on a single exact predicate. In other words, an exact predicate has dimension of Ω_{exact} always equals to one.

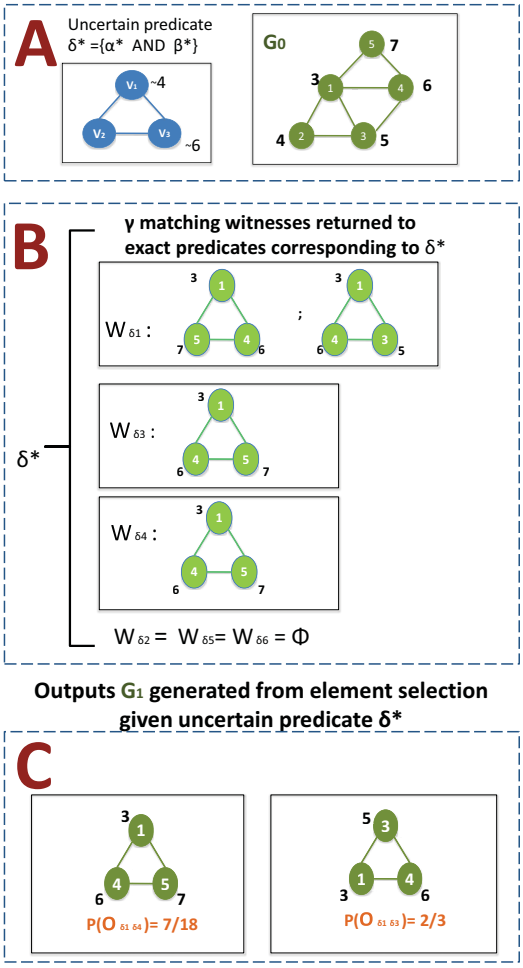

Figure 5.9: This example shows a complicated uncertain predicate structure with two attribute value uncertainties. Part A shows the uncertain predicate and the attributed graph G_0. Part B shows the witnesses generated after the user has input the uncertain probabilities on the exact predicates. Part C shows the final output. The numbers inside the node denote the id. The numbers outside the node denote the age attribute's value. The different colors are used for visualization clarity. The dark green represents input and outputs graphs. The blue represents predicates. And the light green represents witnesses.

5.6 Composition of Workflow

In an analysis, multiple operators are used to manipulate the graph and produce final solutions. In other words, the final outputs are often from the composition of multiple operators, one for each analytic step. Since we have defined the uncertainty model for a single operator step based on either a simple or complex uncertain predicate, we are going

to construct an uncertainty querying model for composition of multiple operators based on our visual analytic algebra framework.

5.6.1 Probability Model for Composition of Operators

Before we discuss how to capture the probabilities for multiple uncertain predicates in an analytic workflow, we define composition. A composition of operators is a workflow which contains multiple operators during an analytic process.

Since our uncertainty model is constructed based on an existing visual analytic algebra framework, which is to be incorporated into visual analytic systems, visualization functions (i.e. force directed layout) are key to the composition of operators. By default, visualization systems will update the view after a user manipulates the data. We assume each operator in the algebra corresponds to a function in the visual analytic system. Therefore, we assume that each operator in the composition is separated by a visualization function. Consequently, consecutive operators cannot be further merged or combined into a single analytic step. Note that a single operator step is a special case of the composition process which contains only one analytic step. As we discussed in Section 5.5, the probabilities of the outputs from a single operator step live in the witness distribution based on the user defined probability set for all exact predicates. Following the same line of reasoning, the probabilities of the final outputs from a composition of multiple operators also live in its witness distribution. In order to be a possible witness for the composition of multiple operators, the witness graphs need to satisfy all of the attribute and structural conditions from all previous uncertain and exact predicates; therefore, the witness distribution at the end of the analysis is the joint probability distribution based on all previous user defined probability sets. Similar to a joint probability distribution based on complex uncertain predicate, the joint probability distribution also is a multivariate distribution. One random variable describes the behavior of one uncertain attribute in predicate structure. Due to a visualiza-

tion call after each manipulation, we can assume independent relations between each of operators in the composition process. We believe that the user independently decides the predicate structure for the problem at hand after obtaining a new visual representation of the data. Based on the independent assumption, we break the joint multivariate probability distribution of the final witness distribution to the product space of all previous user defined independent probability sets. Therefore, we can compute the final output probability for the uncertainty querying problem by maintaining and updating the witness distribution, which is the product of user defined probability sets. Figure **??** shows a workflow of a composition of operators: one aggregation step and two selection steps. The witness distribution for all final outputs live in the product space of these independent components: $\Omega_{A1} \times \Omega_{S1} \times \Omega_{S3}$. If the first selection step is based on a complex uncertain predicate, Ω_{S1} will follow a multivariate distribution. In contrast, if there exists one operator based on an exact predicate structure, the user defined probability will equal to one.

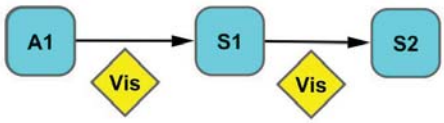

Figure 5.10: Example of a workflow of an analysis with usage of visualization function.

During the composition process, operators cannot be reordered because they are separated by a visualization operator. Witness distributions are updated after each manipulation by multiplying the user defined probability set of the current step. At the end of a given analysis, we can compare and rank outputs based on their probability scores only if the outputs are discovered by using the same sets of operators and predicates.

5.7 Social Networking Application

In order to illustrate the usability of the uncertain predicate framework in a real world setting, we study synthetic social network, Flitter [66]. We demonstrate the effectiveness in reducing users' workload of uncertain querying by dividing the complex uncertain predicate into smaller simple uncertain predicates. Finally, we compare the final outputs between our method to other existing methods.

The Flitter network contains over 6000 user IDs which are represented by nodes in the network. For each user ID, we have information on its user name, city location and sizes of the given location. There are 29876 edges in the network. An edge exists between a pair of nodes or two users if there is at least one communication between them. The goal of this study is to identify a possible criminal network based on a list of given criteria. The suspect criminal network should have total 6 members with name employee, handler (3), middleman and the group leader. Within the their circle, they should form a network structure listed in Figure 5.11.

The user is certain about the connections among all members in the network except the relationship among all three handlers. We use a dashline to represent the uncertain structures. In addition to structural requirement for the criminal network, there are a list of attribute assumptions for each of the criminal members. The assumptions are listed by names in Table 5.7. Based on the output network, we further study the location distribution of the connections of all members in the suspected network. Figure 5.12 shows an overview of the given network in Cytoscape, a visualization tool designs for studying of biological network. At this stage, it is hard to effectively visualize any useful information for the force directed layout.

There are two parts to the problem in Figure 5.13. First, we need to select the suspected

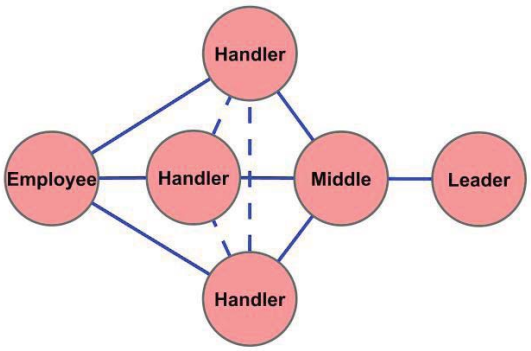

Figure 5.11: This figure shows that the suspect criminal network structure for the Flitter study. Dashlines stand for uncertain connections.

Name	Assumptions
Employee	A user with around 40 connections.*
Handler (3)	Direct neighbor of employee. A user with 30 to 40 connections. Unknown connectivity among three handlers.*
Middleman	Common neighbor of three handlers. A user with around 4 connections.*
The group leader	Direct neighbor of the middleman. A user with over 100 connections.

criminal network based on a list of uncertain queries and visualization function. Then we can aggregate the network and study its neighbors' distribution. Therefore, we can divide the analytic process into two independent components due to the usage of the visualization function. We use the existing Cytoscape function to visualize the data throughout the entire analysis.

The first independent component involves a complex structural selection operator, which contains both attribute and structural uncertainty. Due to the decomposition property of the selection operator, we can further break the selection step into a few conditionally independent components. We begin our first analysis step by selecting all nodes with degree ≈ 40 and its neighbors up to the 7^{th} consecutive steps, since we are interested in uncovering a network of size six and its directed neighbors. In this step, we use an element selection operator, where we select nodes and add a node attribute, $X_{v.label_1}$, for the target node with ≈ 40 condition. With the user defined probability set $\Omega_{\alpha_1^*}$= $(\frac{1}{3}, \frac{1}{2}, 1, \frac{1}{2}, \frac{1}{3})$, we transform this

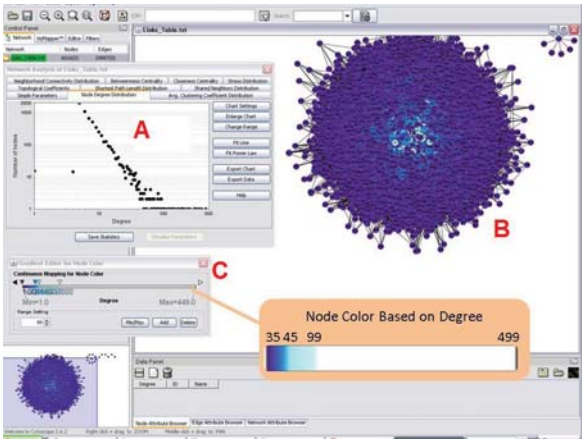

Figure 5.12: Overview of the Flitter network.

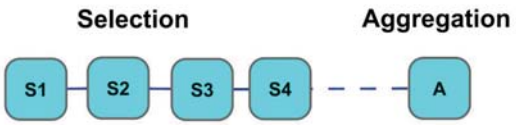

Figure 5.13: Workflow of Flitter study analytic process.

uncertain predicate into a set of exact predicates with degree equal to 38, 39, 40, 41 and 42. Since the given assumption indicates the suspected employee should have around 40 connections, the user defines higher probabilities to predicates with degree requirements close to the given assumption, and a probability of zero to any predicates with 3 degrees away from the given assumption. As listed in section 5.5, $\Omega_{\alpha_1^*}$ is only a probability set. Each element of the defined probability set follows an independent distribution which describes how well the exact predicate structure captures the given target structure. After the first element selection step, there are 26 output graphs with different output probabilities. All output probabilities are constructed based on the witness distribution which equals to the user defined 5 probability distributions, $\Omega_{\alpha_1^*}$. By requiring user input probabilities in the predicate step instead of the output step, we reduce the user's workload from attaching probability scores to all 26 outputs to only define 5 different predicate requirements.

Based on the selected output, we are going to select output graphs containing 3 directed neighboring nodes of employee and with degree between 30 to 40. Since the user believes that three handlers work in parallel on separate cases, they are less likely to have connections among three of them than have no connections at all. Based on this assumption, the exact predicate set contain only two exact predicates. They are $\alpha_{2.1}$ with no connection $(P(\alpha_{2.1}) = 1)$, $\alpha_{2.2}$ with only one connection$(P(\alpha_{2.2}) = \frac{1}{2})$. $\Omega_{\alpha_2^*}$=$(P(\alpha_{2.1}), P(\alpha_{2.2}))$ contains only conditional probabilities, for example,$P(\alpha_{2.1})$ is the probability of no connections among all three handlers given the selected employ from the previous step is a correct suspect. Because user applies the set selection operator here, one selects the entire witness graph instead of only the matching elements. Moreover, we also use the composition function to create a new node attributes for the selected handlers. At the end of this step, the size of output list is further eliminated to 9 outputs and the probability attached to each output is updated based on the current witness distribution. The current witness distribution lives in the joint probability space of $\Omega_{\alpha_1^*}$ and $\Omega_{\alpha_2^*}$, since all witnesses from the current step are products of two consecutive selection steps.

The next step is to select the middle man who is the common neighbor of 3 handlers with degree ≈ 4. Similar to the first selection step, the exact predicate set contains predicates have the same structural requirement, but different degrees ranged from 2 to 6 with probability set $\Omega_{\alpha_3^*}$. In this step, the user picks the probability set $\Omega_{\alpha_3^*}$ = (1,1 ,1, $\frac{1}{2}$, $\frac{1}{3}$), since one believes that the middle man is more likely to be a less active user than highly connected user. Once again, user also uses set selection to select outputs. At the same time, the witness distribution is updated once more to $\Omega_{\alpha_1^*} \times \Omega_{\alpha_2^*} \times \Omega_{\alpha_3^*}$, where $\Omega_{\alpha_3^*}$ is conditional independent from $\Omega_{\alpha_1^*}$ given $\Omega_{\alpha_2^*}$. By adopting a conditional independent assumption amongst all consecutive analytic steps, our framework effectively decreases the user's workload in defining joint witness distributions.

The final step of the first independent component is to discover the leader of the criminal network. Based on the given assumption, the group leader should be a highly connected nodes with degree over 100. This step is a set selection operator based on a exact predicate structure. The exact predicate query is treated as a special case of the uncertain problem in the analytic process with one exact predicate structure in the set with probability one. After all selection steps, only 2 outputs in Figure refcrime1 left with different output probabilities based on witness distribution, $\Omega_{\alpha_1^*} \times \Omega_{\alpha_2^*} \times \Omega_{\alpha_3^*}$. We rank all output based on output probabilities. The user can use the visualization function to further eliminate the size of output list if needed. Since there are only two output graphs left in this study, we keep all of them for the next step.

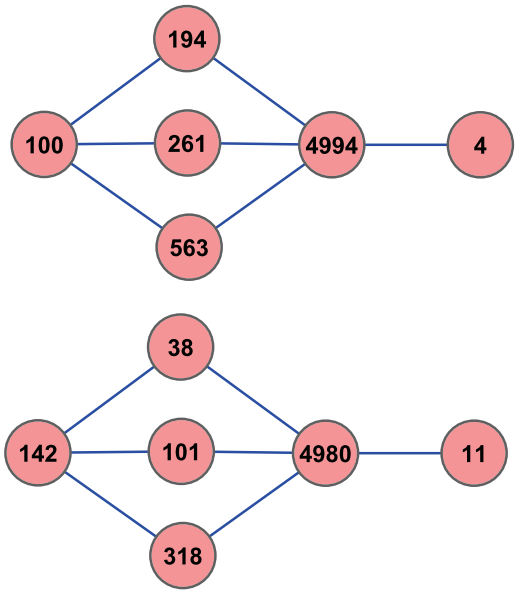

Figure 5.14: Two output graphs from the first selection component. Number in each node is the user ID. The output probability for the upper graph is $\frac{1}{2}$ and the output probability for the lower graph is $\frac{1}{3}$, because of the degree differences between two middleman, user 4994 and 4980.

Selection Step:

1. Element selection

   α_1^*=$\{\{v_1, \ldots, v_7\}$,$\{e_{12}, \ldots, e_{67}\}$, $X_{v1.degree} \approx 40\}$

Exact predicate set for α_1^*:

- $\alpha_{1.1}$=$\{\ldots, X_{v1.degree} = 38\}$
- $\ldots$
- $\alpha_{1.5}$=$\{\ldots, X_{v1.degree} = 42\}$

with probability set $\Omega_{\alpha_1^*}$=($\frac{1}{3}$, $\frac{1}{2}$,1,$\frac{1}{2}$, $\frac{1}{3}$).

2. Set selection

   α_2^*=$\{\{v_1, \ldots, v_4\}$,

   $\{e_{12}, e_{13}, e_{14}, \approx e_{23}, \approx e_{34}\}$,

   $X_{v1.label_1}$ ="Y",$X_{v2-4.degree} \in [30, 40]\}$

   Exact predicate set for α_2^*:

   - $\alpha_{2.1}$=$\{\{v_1, \ldots, v_4\}$,$\{e_{12}, e_{13}$,$e_{14}\}$,$\{!e_{23}$,$!e_{34}\}, \ldots\}$
   - $\alpha_{2.2}$=$\{\{v_1, \ldots, v_4\}$,$\{e_{12}, e_{13}$,$e_{14}, e_{23}\}$,$!e_{34}, \ldots\}$

   with $\Omega_{\alpha_2^*}$=$(1, \frac{1}{2})$.

3. Set selection

   α_3^*=$\{\{v_1, \ldots, v_4\}$,$\{e_{14}$,$e_{24}, e_{34}\}$,

   $X_{v1-3.label_2}$ = "Y", $X_{v4.degree} \approx 4\}$

   Exact predicate set for α_3^*:

   - $\alpha_{3.1}$=$\{\ldots$,$X_{v4.degree} = 1\}$
   - $\ldots$
   - $\alpha_{3.6}$=$\{\ldots$,$X_{v4.degree} = 6\}$

with $\Omega_{\alpha_3^*}$=$(1, 1,1,\frac{1}{2}, \frac{1}{3})$.

4. Set selection

   α_4=$\{\{v_1, v_2\}, e_{12}, X_{v1.label_3}$="$Y$",

   $X_{v2.degree} > 100\}$.

For the second part of the analysis, where one is interested in studying the location distributions of the suspected criminal network and its neighbors, the user can use the visualization function to check the location distribution of neighbors for each of the members. In order to have a better understanding of the overall distribution of all neighbors, we use the element aggregation operator to aggregate all members of the network together to a super node and call a summary composition function to compute the location distribution of neighbors of the super node. Since this step is constructed based on an exact predicate, the witness distribution stays unchanged from the network structure to the super nodes.

5.8 Conclusions

In this chapter we presented a probability model for uncertainty in queries. We incorporate this model into a visual analytic algebra. We present multiple examples on how it can be used. We also show how the probability can be propagated in a workflow of uncertain predicates. We would like to test this system with real users and situations to explore its benefits. There are still many areas of uncertainty in graphs to explore. The next step would be to incorporate uncertain predicates to an uncertain graph model. Another challenging problem is how to maintain the lineage of the different uncertainty values created in a workflow.

CHAPTER VI

Missing Attributes

6.1 Introduction

The study of complex networks in biological, social, and other applications has received interest in recent database literature. High-throughput experiments and the integration of different data sources create these complex networks. A major impediment to analyzing these networks is incomplete data. There are many situations in which it would be useful to answer questions such as "What is the functional annotation of this protein?" or "Is there a hotel in this neighborhood for under $100 per night?" when this information is unavailable. Furthermore, incomplete data greatly affect the performance of graph analysis. To obtain meaningful results from analysis, we must resolve this missing information first.

In terms of the thesis $overview\ example$ of cell phone network analysis, [46], which includes 400 unique cell phone call records over a ten-day period, the missing attribute information can be studied over a geographical analysis of calls. Each call record includes the satellite which was used to make a call. However, we can present a possible missing attribute scenario if we imagine the collection of phone call data is integrated from multiple phone networks. In this case, maybe the satellite information might not be available for some nodes, if this information is not shared by one of the integrated networks. We can interpolate these missing values by using the predicting missing attribute algorithm

described in this chapter.

In this chapter, we study how to resolve such missing information. We focus, in particular, on missing attributes at nodes in the graph. Predicting values for attributes is a rapidly emerging area of graph interpolation. Predicting values not actually recorded in the data set involves guess-work, of necessity. The general idea is to look at attribute values in the neighborhood of a query node n and use frequently occurring values to determine the attribute values for n. In some fields, such as Bioinformatics, this class of techniques has even been given a name, Majority, because it is used so frequently. A major shortcoming of such techniques is that they ignore network properties and structure.

This technique assumes highly assortative networks, which means similar nodes are more likely to interact with one another. In practice, many interpolation algorithms need to tackle problems where the network's mixing patterns are heterogeneous with rich and diverse interactions. Many different levels of interactions exist for different reasons. For example, biologists describe symbiotic relationships varying from parasitic, communal, or mutual. Attributes of nodes in such relationships would have complementary values rather than being the same. For instance, an E-bay bidder searching for a vintage poster would interact with a seller of these posters. Bidders and sellers usually do not have similar characteristics. Of course, there are other relationships where we expect participating nodes to have similar attribute values. A complex graph may frequently comprise both types of relationships. Furthermore, even within a single relationship, some attributes may be similar and others complementary. For example, individuals may typically date others with similar education level but opposite sex. These interactions and other types of interactions blend into one complex graph. The "guilt by association" approach is too rigid to interpolate such complex graphs.

Figure 1 shows a protein-protein interaction sub-graph extracted from the earthworm's

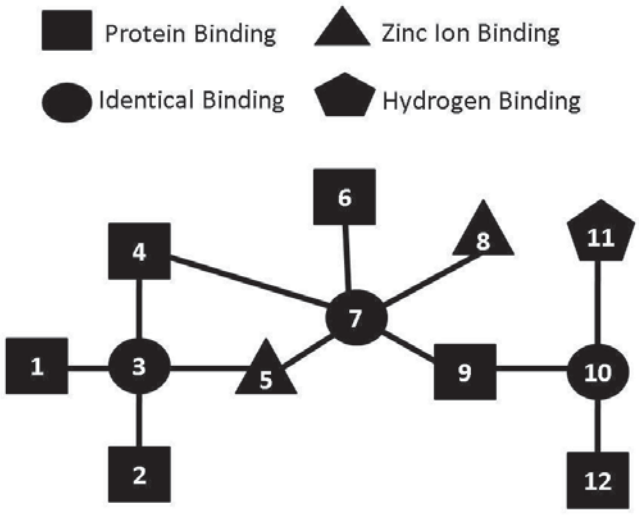

Figure 6.1: Protein-Protein Interaction network.

proteome. Within large protein-protein interaction networks, patterns such as this small subgraph exist. Proteins are the uniquely labeled nodes, interactions are edges, and node shapes denote the protein's functional characteristic. These graphs are used to study protein interactions which lead to new discoveries in biological cells and diseases. Interpolation is used to determine the missing annotations. In this example, we see circle annotations interacting with the other shapes, but not with other circles. Based on this subgraph, square annotations are most abundant. In this graph we can take an example of a missing annotation we would like to interpolate. Let us assume that the protein labeled 3 is missing its annotation. If we only take into account annotation frequencies, we would mistakenly guess protein 3 is a square. Most current interpolation methods would also incorrectly interpolate a square for protein 3. The weakness with these approaches is that they do not adapt to the structural features of the graph. In this chapter, we open a new direction of attack on the interpolation problem. The natural next consideration is the relationship between interactions and attributes. Protein 3 interacts with squares and triangles, and this subgraph shows that squares and triangles interact with circles. Therefore, when

we consider relationships, we correctly interpolate a circle for protein 3. We present a novel algorithm that analyzes the surrounding local wiring, finds interesting patterns, and considers disassortatively mixed information.

The key idea underlying our approach is that there exists a correlation between the attributes of a node and its interactions. By understanding these correlations that exist in the graph, we can interpolate missing values. In our algorithm we explore interaction pair similarities to interpolate missing attribute values. Our hypothesis is that looking at the interactions of the query node will give us a much better estimate of its attributes than simply by looking at its neighborhood.

We present two approaches of single attribute prediction. Our first method compares the equivalence of a query interaction pair with neighboring interaction pairs. Based on this comparison, we return values for the missing attribute. The second method uses Bayesian inference analysis. We collect the probabilities of the values occurring in the neighboring interaction pairs. Then we use these probabilities to determine the most likely values for the missing attribute. We demonstrate the effectiveness and broad applicability of our new interpolation approaches using extensive experiments on real data sets. We explore graphs from a variety of applications including biological, trade, and social networks. These graphs have different network properties, attributes, and interactions. In each case, our method gives higher precision of the results. We also compare each methods sensitivity to graph perturbations. The results show that our method is a more effective interpolator compared to current state of the art methods. Finally, multiple nodes in a graph may be missing values. We present both parallel and sequential methods for multiple node interpolation.

In the following sections we will discuss graph interpolation and our approach. First, we review related work specific to this chapter. Section 2 we will define the problem

precisely. In Section 3 we will cover the interpolation algorithm. Section 4 we discuss our experimental results including a review of the current state of the art interpolation methods. We conclude in Section 7 with summary and possible future directions for our work.

6.2 Related Work

There are several studies on attribute prediction for protein interaction graphs. Previous methods operate on a single node or more precisely the missing attribute at a single node [14], [15]. In these single point cases, the interpolation algorithm canvases the neighborhood for similar points and returns the most commonly occurring value. Our work introduces the interaction pair for interpolation.

Chua et al. [7] study on protein interaction graphs show that proteins have significant functional associations with their level-2 neighbors. In our work, we apply indirecting function associations in a greater neighboring region. Functional Flow [13] is network-based algorithm which simulates flow between nodes over d number of time steps. Similar to Functional Flow, our work incorporates network information to make predictions.

Also, there has been significant effort in quantifying the similarity of vertices. Leicht et al. [16] define a recursive similarity based on the concept that two vertices are similar if their immediate neighbors in the network are similar. We utilize a similar notion in our simple interpolation algorithm when we compare two sets of interaction pairs. If the neighbors of two nodes are similar, then the nodes are similar. Our work is also related to interpolation for non-graph data. In sensor data, the MauveDB project [17] provides users and application developers statistical models for interpolation of missing values. However, we develop methods for graph data. And more generally there are several studies on graph problems including summarization [18], identifying subgraphs that connect multiple query

nodes [19], [20], identifying communities [21], and indexing graphs [22]. However most previous works ignore correcting for missing values in a graph.

6.3 Problem Definition

We are given an undirected graph $G = (V, E)$ with vertex set V and edge set E and associated with each vertex is an attribute vector X. Each vertex is also labeled with a unique name (or ID) for identification purposes. Attributes are single valued and may be categorical or numerical. An example of a categorical value in a social network is a person's job title. The attribute vector X has missing values. Our goal is to populate the missing values in X such that the final graph produced is an accurate representation of the complete graph. In Section 4 we describe our experiments and metrics we use to determine if the final interpolated graph is an accurate representation. To initially make the problem space more concise, we restrict the operation to interpolate a given missing attribute for a single node. Later we consider the extensions necessary for multiple missing attributes and nodes.

6.4 Interpolation Algorithm

Given a network and an identified query node, as shown in Figure 2.1, our first step is to find all interaction pairs in which the query node participates. In this fictitious example, the shapes represent attribute values at the node. The network contains one attribute value. The query node containing the missing attribute is denoted with the question mark. We find two types of query interaction pairs because the query node interacts with nodes with either square-valued or triangle-valued attributes, shown in Figure 2.2. We then look in the neighborhood of the query node to find other interaction pairs that could match with the two interaction pairs of interest to us. The neighborhood is a parameter to be specified. In Figure 2, the neighborhood includes the entire graph. In Figure 2.3 the arrow points to

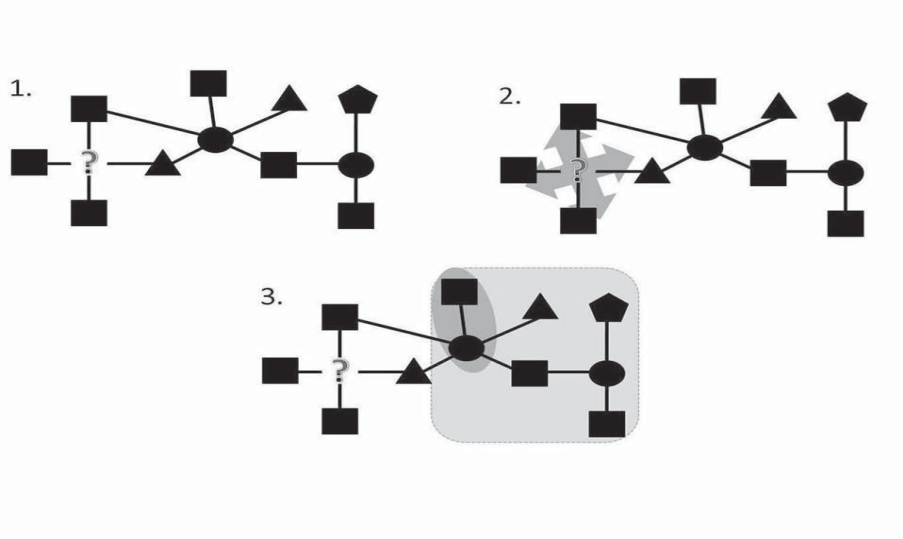

Figure 6.2: Overview of interpolation. At stage 1 we see the query attribute denoted by the question mark. In stage 2 we collect the interaction pairs with the query node and its direct neighbors. The new interaction pairs on the side are the query interaction pairs. Next we collect the neighboring interaction pairs, stage 3. We continue in the simple interpolation method by comparing to these interaction pairs for similarities with other attributes. In the Bayesian based interpolation method we collect the probabilities of the values occurring at these interaction pairs and use these probabilities to determine the likelihood of a value for the query attribute.

the interaction pairs found within the neighborhood.

To assign an appropriate value to the missing attribute, we must effectively analyze and compare to these interaction pairs. We present two methods: simple interpolation and Bayesian based interpolation. In the simple interpolation method we perform the traditional frequency based methods which in the past operated on a single attribute unit. However now we weight each value found with the number of attribute values in common between the query and neighboring interaction pairs. We also present a Bayesian based approach because of the multi-dimensionality of the data, Bayesian inference analysis will naturally find correlations and overlaps between attributes. Effectively multiple characteristics are considered simultaneously. Next we must compare the query interaction pairs found in Figure 2.3 with the inferences made. For each possible value of the missing attribute, we determine its likelihood considering the other attributes in the interaction pair

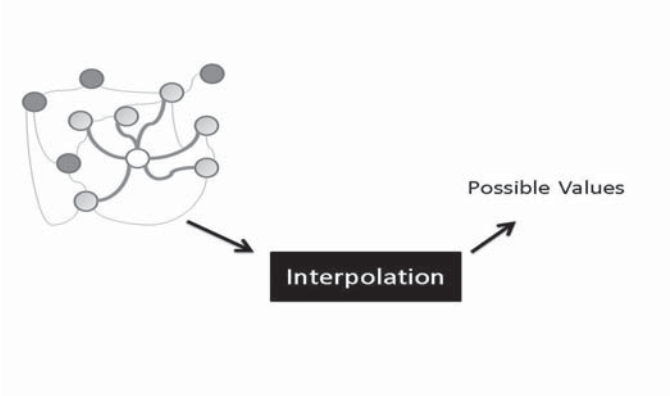

Figure 6.3: Input and outputs of the program shown. A sample input graph is also shown. The white node represents the specified node with the missing attribute value and the bold edges represent a radius of one away from the query node to use during interpolation. The shaded nodes within this radius of one to use during interpolation, in this case all six nodes are selected.

and the inferences made from neighborhood interactions.

Many important issues were glossed over in the overview presentation above. In the next subsections, we work through these in detail, focusing on a single identified query node and the prediction of values for a single attribute at that node.

6.4.1 Collection of Resources

Given the input node, the algorithm first defines the ***query interaction set***. The query interaction set consists of interaction pairs in which the query node participates.

Before interpolation, the following must be specified: the radius of the *r* nearest neighbors to consider in the neighborhood, and the maximum number of *k* candidate values to produce for each missing attribute's value. The parameter *r* is user-specified. We crawl the graph within a radius of *r* to collect neighboring interaction pairs. A radius of 2 would mean use neighbors up to two edges away from the query node. We use edges as a measure of distance in the graph. In Section 4, we will describe how to choose *r* for a given problem. Here, we just assume it has been specified.

The number of values for an attribute, k, is an application dependent parameter. We

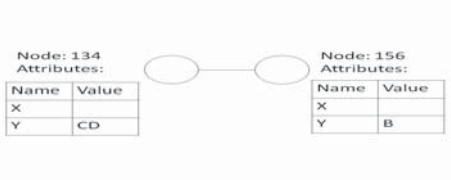

Figure 6.4: An example interaction pair between nodes 134 and 156. Both nodes have no values for their X attribute.

may often know that the attribute of interest is unique-valued, then k is 1. If there is no application constraint on the number of values for the attribute, then k is set to infinity. Note that the actual number of values that the algorithm will predict is less than or equal to k, so setting it to a high value does no harm. Typically attributes are single valued but we provide this option for generality.

Figure 3 shows the high-level input and outputs for interpolation program. Figure 3 also shows an example input graph where the white node represents the node with the missing attribute. The bold edges represent a specified radius of 1 to use as the local region during interpolation. The shaded nodes represent the total nodes within this radius of 1 to use during interpolation.

6.4.2 Analysis Method

Given a query interaction pair and a set of neighborhood interaction pairs, we show how to determine values for the missing attribute. We have three decisions to make. First, interaction pairs must be aligned before comparison. Second, we must appropriately exploit the extra information attained from the interacting partner and the other attributes of the query node. Additionally, we must handle partially annotated information. The graphs may have a lot of missing information, creating error in alignment and difficulty in applying the extra information from the interacting partner. Figure 4 shows a partially annotated interaction pair where the attribute X of both the nodes is missing values but attribute Y has values. Finally, we decide how to compare interaction pairs, where we demonstrate a

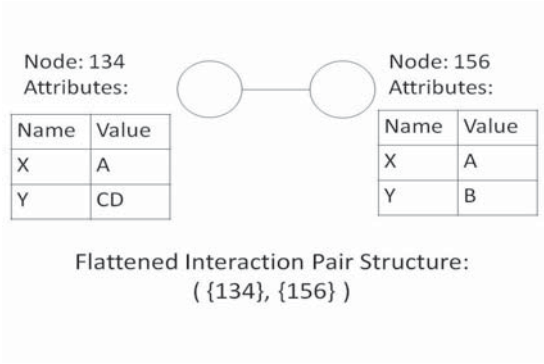

Figure 6.5: The interaction pair has been arbitrarily flattened before alignment.

simple and a Bayesian approach. We describe these through steps in turn.

Alignment

To assign a direction to the interaction pair, we flatten these structures so that one of the nodes is considered first and the other is the second node in the interaction pair. Then we can align in a consistent fashion and access an attributes value by referring to either First.(node level information) or Second.(node level information) in the interaction pair. Basically, a flattened interaction pair structure is an ordered list of the nodes. The exact order is determined by alignment. Figure 5 is an example of a flattened interaction pair structure before alignment has assigned the permanent direction.

With flattened interaction pair structures it is possible to complete alignment. We assume no hierarchy exists between the attributes, nor do we have any prior application dependent information on attribute value ordering; therefore, we apply lexicographical ordering. We make further assumptions about the attributes existing at nodes. First, we assume that all nodes have the same set of attributes. This assumption is reasonable because we are only broadening the scope of what the original node entity describes; hence, not losing any information. The drawback of generalizing nodes is we may compare two formerly different types of nodes. However, the comparison stage corrects for this possi-

bility. We also create an empty value and assume it is ordered earliest lexicographically to cover cases of partially annotated information.

Now in alignment we determine whether to place the nodes in the interaction pair in first or second position based on the attribute values at each node. For each node in the pair, first we order the attributes, since each attribute has a name. Then we compare the two nodes ordered attributes at their values to determine the interaction pair direction. Algorithm 1 describes the alignment process for the interaction pairs after empty valued placeholder insertion.

Algorithm 1: Alignment

Input: Interaction Pair
Output: Aligned Interaction Pair
foreach *Node* **do**
    Sort attributes by name in lexicographic order.
    while *Node order not determined* **do**
        Compare next attribute's value between nodes.

Again, we only align the neighborhood interaction pairs, not the query interaction pair. This provides a consistent alignment to compare against the query interaction pair, as well as for efficiency so we don't have to test all directions. Therefore, the final alignment for the interaction pair in Figure 6.4 is $\{156\}, \{134\}$).

Simple Interpolation

We describe first a preliminary approach to finding values given a query interaction pair and a set of aligned neighboring interaction pairs. Next we show our Bayesian based method. Suppose we have the toy example query interaction pair and two neighboring interaction pairs shown in Figure 6.6.

In this example, we try to interpolate a value for the question mark in attribute X of the query interaction pair. The neighboring interaction pairs are aligned. We do not assign a permanent direction to the query interaction because the missing value might affect the

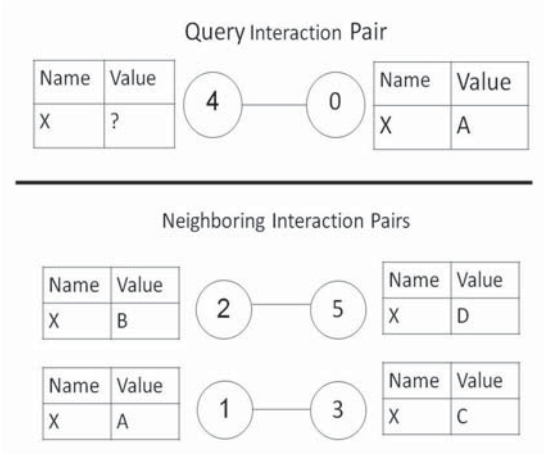

Figure 6.6: The query interaction pair is between nodes 4 and 0. The value we interpolate for is node 4's X attribute. There are two aligned neighboring interaction pairs. We compare the both directions of the query with each neighboring interaction pairs. In this case, the interaction pair between nodes 1 and 3 has more similarities, namely a node with an A valued X attributed, with the query versus the interaction pair between 2 and 5. Therefore, the simple interpolation algorithm predicts C with the highest rank for the query attribute.

direction. Instead we compare the query with the others using both possible directions. Without the aid of application-dependent comparison information, a simple comparison method is equivalence. If the query has equivalent values for the other attributes in a neighboring interaction, we copy the neighboring interaction's attribute values to the query attribute. In the example shown, the missing attribute in node 4 would acquire the value *C* for its missing attribute. The interaction pair ($\{2\}$, $\{5\}$) has no similarities with the query. However, the interaction pair ($\{1\}$, $\{3\}$) when compared with the query aligned as ($\{0\}$, $\{4\}$) has an equivalent attribute value. Finally, we use the equivalence as a weight and return the top p candidates. Algorithm 2 shows the straight forward interpolation method for interaction pairs.

Essentially Simple Interpolation performs a Majority-Vote method on an interaction pair unit, instead of the attribute at a single node. However, strict equivalence may produce incorrect results when there are several attributes and several missing values.

We need to first examine correlations between attributes to increase the match-ability

Algorithm 2: Simple Interpolation

```
Input: Query Interaction Set, Aligned Neighboring Interaction Pair Set
Output: Predicted Values for Missing Attribute, a
Create the results list of value score pairs;
foreach Query Interaction Pair, q do
    for Both Directions of q do
        foreach Neighbor Interaction Pair, n do
            if n is nonempty for a
            score ← |X_q ⋂ X_n|
            Update results with new value and score
Return top k values from results
```

of the query with neighboring interaction pairs. So we present a Bayesian-based approach to find correlations between attributes to effectively consider multiple characteristics simultaneously.

Bayesian Based Interpolation

We use a Bayesian based method to infer values for missing attributes. We collect probabilities of the values occurring in the neighboring interaction pairs. Next, given the values occurring in other attributes in the query interaction and the collected probabilities, we return the most likely value for the missing attribute. We apply Bayes' theorem:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)} \tag{6.1}$$

Where *H* represents the possible value, *E* represents values present in other attributes in the query interaction. After computing likelihood for all possible values for the attribute, we return the top *k* resulting candidates. An obvious limitation of this method and the simple interpolation introduced above is that only values observed in the graph are possible interpolation candidates. In other words, we do not infer or create new or unseen values during interpolation. So, we do not require any application dependent instruction. Also, for both approaches, if values are spread uniformly in the graph and independent on

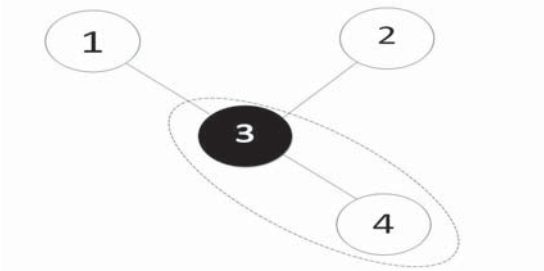

Figure 6.7: Though these two subgraphs are trivial, they show an example where pairwise interpolation subsumes the majority vote based on single nodes when the values are distributed evenly in the graph. In this case pairwise interpolation is possible in both networks a and b; whereas, majority-vote for single nodes is only possible for network b.

interactions of other attributes, then our methods degenerates to a single node Majority-Vote. Basically our method subsumes a single node Majority-Vote solution as can be seen in Figure 6.7. Although the subgraphs in the Figure are trivial, pairwise interpolation is successful for both networks.

Now we discuss which probabilities we collect and how we use these probabilities for interpolation. After alignment, we collect the prior and conditional probabilities of the values occurring in the neighboring interaction pairs. The priors for each node in the interaction pair, either the first or the second, has its own separate probability table. We define the conditional probability as the probability of seeing a value given a different attribute value occurs simultaneously. These attributes can be in the different nodes of the interaction pair, or even within the same node. Algorithm 3 outlines the probability collection process described above.

Algorithm 3: Probability Collection

Input: Aligned Neighboring Interaction Pair Set
Output: Prior and Conditional Probability Tables
Create first node prior table, **first**;
Create second node prior table, **second**;
Create conditional probability table, **conditionals**;
foreach *Interaction Pair, **I*** **do**
    Update **first** and **second** with values in **I**;
    foreach *Attribute value pair in **I*** **do**
        Update **conditionals**;

The next step is to use these collected probabilities to determine a value for the missing attribute. We iteratively apply Bayes' theorem. We use one piece of attribute value evidence to calculate the probability and then use the probability as a new prior probability to calculate the next probability with the next piece of information. For example, below is the iterative Bayesian theorem to apply additional attribute value probabilities:

$$P(v|E_1 \cap E_2 \cap ... \cap E_N) = \frac{P(E_1|v)P(E_2|v)P(v)}{P(E_1|v)P(E_2|v)P(v) + P(E_1|\neg v)(E_2|\neg v)P(\neg v)} \quad (6.2)$$

In the equation, *v* is a possible value and E1, E2, ... En are other attribute values. Similar to the simple interpolation, we predict values for the missing attribute using both alignments of the query interaction pair. We cannot assume a single alignment because the value at the missing attribute can sway the alignment direction. Comparing with both directions may seem to be conflicting with our decision of inserting empty values during alignment. However in interpolation we do not return empty values for missing attributes. If empty value is truly a possibility, we differentiate between "truly empty" and "missing empty." A "missing empty" value is created during alignment execution and "truly empty" is a graph attribute value. Therefore, it is possible to interpolate a "truly empty" value for an attribute. We compute the probability for all possible values of the missing attribute for both alignments of the interaction pair. Next we rank these values by their total probabilities to return the top **k** candidates. Algorithm 4 outlines the process described above.

6.4.3 Pair versus Partner

In our current interaction pair interpolation model we use a one-at-a-time approach to matching interacting pairs with neighborhood information. In other words, if a query interacts with several nodes, we interpolate a value considering each interacting node with

Algorithm 4: Bayesian Based Interpolation

```
Input: Query Interaction Set, Probability Tables
Output: Predicted Values for Missing Attribute, a
Create the results list of value score pairs;
foreach Query Interaction Pair, q do
    for Both Directions of q do
        foreach Possible Value, v, of a do
            Compute score = P(v) using Equation 2
            Update results with new value and score
Return top k values from results
```

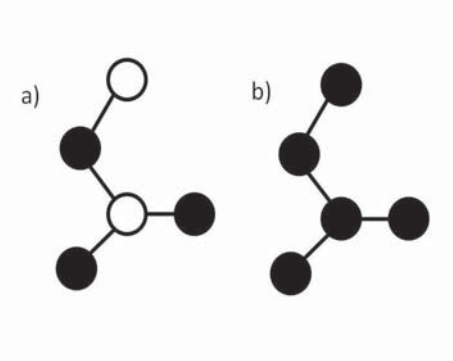

Figure 6.8: In the figure above, if node 3 is the query node, then nodes 1, 2, and 4 are the interacting nodes. In an interacting nodes approach, only the white nodes will be used during prediction. An example interaction is circled. In a one-at-a-time interaction pair prediction approach, each query interaction pair is separately compared with neighboring interaction pairs.

the query node separately. We accumulate the values from each interaction pair at the end. However, there are other possible methods within the realm of pair wise interactions. We elaborate on the differences between different methods and their affect on the interpolation algorithms stated. For example, interpolating by considering only the interaction nodes is a possibility. Our interaction pair structures use both the query and interacting node. Figure 6.8 shows what we mean by interacting node and interacting pair. If node 3 is our query, then nodes 1, 2, and 4 are its interacting nodes. The interacting pairs of node 3 are ($\{1\}$, $\{3\}$); ($\{2\}$, $\{3\}$); and ($\{4\}$, $\{3\}$).

The benefit of excluding the query node is smaller information storage and analysis. We can simply modify our algorithms above by ignoring the query node's attribute to fit this model. The disadvantage of this method is we lose potentially useful information from

the query node.

The opposite extreme of our method is to consider interaction pairs all together. For example, in Figure 6.8, we interpolate at once with query interaction pairs ({1}, {3}); ({2}, {3}); and ({4}, {3}) by examining the neighborhood for similar groups of interaction pairs. A conceptual argument for using the interacting partners at once is that interacting pairs cannot be thought of existing in isolation, just as nodes cannot be thought of existing in isolation within the graph. An interacting partner subunit in the graph can similarly be thought of existing with its connected interaction pairs. By considering interaction pairs separately, we may be giving uneven consideration when they are really co-existing pairs of the node. **Insert** sentence here on an example of why exactly it is different from the separate case. Unfortunately, the disadvantage of this method as we will see in our experiments is that there are fewer examples of larger subunits in the graph. A compromise between both methods could be to use some of the interacting pairs together; not necessarily all together or all separately.

6.4.4 Multiple Attributes and Multiple Nodes

Above we described interpolation for one attribute on one node at a time. In many cases, we will have an entire graph with several missing attributes at several nodes. For example, a protein-protein interaction graph for the yeast organism is missing 1/3 of the proteins' functional annotations. Therefore, now we describe how to extend this method to multiple attributes and multiple nodes. Without little modification to the algorithm above, to interpolate multiple attributes at the query node, we predict values for each attribute with missing values. We use the same neighboring interaction pairs and probabilities collected for each attribute in the same query node. In fact we can predict values for attributes with values as well. But currently, we do not use the existing values in the attribute we interpolate in the Bayesian step as evidence for the same attribute. We return the top k

candidates excluding those values already present in the attribute.

We present two methods for interpolating multiple nodes, for example an entire graph. Option one is to interpolate each node independently and in any order - in a simple parallel method.

Algorithm 5: Graph Parallel Interpolation

Input: Graph
Output: Interpolated Graph
foreach *Node* **do**
    Call Interpolation
    Save predicted values in separate value set

The second method is to serialize the process. Serializing the interpolation process causes past node interpolations to affect the outcome of future node interpolations. The sequence of nodes can be in node increments or in groups of nodes. For example, for the next group of nodes, we utilize the previously predicted values as evidence. But the nodes in the same batch use parallel interpolation.

Serialization requires determining an appropriate sequence to use. One possibility is to use a sequence of decreasing important nodes. In complex network analysis, there are several definitions of important nodes. Moreover, a node's importance depends on the application. However, some common characteristics used to describe nodes include degree and betweeness. In terms the modifications required in Algorithm 5, we simply replace parallelize with a priority queue and allow interpolated values as evidence.

A possible weakness of serializing is we may percolate errors if one predicted value is incorrect. On the other hand, if predicted values are correct, there is more evidence for better future prediction. Another argument for this method is that the value we choose for one node might depend on how we interpolated another node. In our experimental section we examine both parallel and sequential interpolation methods for graphs.

Table 6.1: Data Set Network Properties

Data Set	Nodes	Edges	Radius	Clustering Coeff.	Avg. Path Length	Avg. Degree
Yeast Protein	1884	x	x	x	x	x
World Trade	80	875	6	0.55	2.23	21.87
Hollywood Film	102	192	10	0.0	3.751	3.80

6.5 Experimental Results

In this section we describe the data sets and experimental methodology we use to demonstrate improvement our algorithm has over the current state of the art. We also present our results.

6.5.1 Data Sets

We selected a diverse set of applications to interpolate. Table 1 shows the various network properties calculated for the different data sets.

The three data sets we use are the following:

We take one data set from the Yeast Protein-Protein Interaction data set [7]. Interaction detection methods have led to the discovery of thousands of interactions between proteins. Protein-protein interactions are key determinants of protein function. This correlation is useful because many proteins are missing annotation. We use the annotations from the Munich Information Center for Protein Sequences (MIPS) [8]. The attribute we interpolate is function which has 13 possibilities. There are 1884 proteins in this data set.

The second data set is world trade in miscellaneous manufacturers of metal in 1995, among 80 countries [9], [10]. The attributes in this data set consist of total import, total export, and a countrys world system position (core, strong semiperiphery, weak semiperiphery, periphery) according to Smith and White analysis [11]. Here, we predict the world system position of a country.

The next data set is from Hollywood film and music collaboration during 1964-1976 [9], [12]. The network contains the collaboration of 40 composers of film scores and 62

producers who produced a minimum of five movies in Hollywood during 1964-1976. The attribute we interpolate is the profession of a person in the network. Either a person is a producer, a top 5 composer, or a composer but not top 5.

6.5.2 Experimental Methodology

We give a short description of the other algorithms we use for comparison. Then we describe our experiments.

Other Interpolation Algorithms

We implement the majority method described in [13]. Majority examines the neighboring vertices within a specified region to find the most frequently occurring values for the missing attribute. Next it reports the top N frequently occurring values. Although this method is simple, it compares competitively with other more recently proposed interpolation methods [5], [7]. Therefore, we use this as our baseline to determine improvement.

The second method, Functional Flow, [13] is network based algorithm which simulates flow between nodes over d number of time steps. Nodes initially have infinite potential for attribute values and zero otherwise. The attribute values are then iteratively spread from nodes with higher potential to their immediate neighbors with lower potential. At each iteration they calculate the flow which has entered the nodes. Flow is limited based on the node capacity. They implement these set of flow rules which incorporates network structure into their interpolation algorithm.

Effectiveness: Precision

We test the performance using a two-fold cross-validation carried out by dividing the data set into two groups, and each group in turn, is separated from the original data set and used for testing. The goal of each method is to interpolate the attribute value of the nodes in the test set using the remaining nodes in the graph. We measure the quality

of an interpolation of the overall graph with precision. The precision measures the total number of attribute values predicted correctly for each missing attribute over the total of attribute values predicted for each missing attribute. The recall measures the total number of attribute values predicted correctly for each missing attribute over the total number of known missing values for each missing attribute. Below is the definition of precision:

$$\sum_{\text{interpolated attributes}} \frac{\text{num of values correctly predicted}}{\text{num of values predicted}} \tag{6.3}$$

This metric will give us the overall effectiveness of an interpolation algorithm and is a common metric used to determine the usefulness of an interpolation algorithm. Recall is often another measure for prediction. The recall measures the total number of attribute values predicted correctly for each missing attribute over the total number of known missing values for each missing attribute. The following is the definition of recall:

$$\sum_{\text{interpolated attributes}} \frac{\text{num of values correctly predicted}}{\text{num of values missing}} \tag{6.4}$$

The recall results are trivial for these data sets because we only interpolate single valued attributes.

For each data set we average the results over 30 runs. Figure 9 shows the precision results comparing Majority, Pair, and Functional Flow for the Hollywood Film Collaboration data set. Pair is our Bayesian parallel interpolation method.

This data set is a small social network of 102 people. There are two distinct groups of people: composers and producers. There is collaboration across groups but none within the groups. Therefore, we can see from its clustering coefficient of 0 in Table 1 that this data set is bipartite.

However, this data set is not trivial because the composer group is divided into two groups: top five composer or not a top five composer. Overall we see that Pair outperforms Majority and Functional Flow. Up to a radius 3, Pair significantly outperforms Majority and Functional Flow. Beyond a radius 3, Majoritys precision improves and reaches a plateau of about 0.4. As the radius increases, there are more values which allows Majority to randomly make correct predictions. The ratio of composers to producers is 40:62. Unlike Majority, Pairs precision decreases as the radius increases. As we move further from the query node, the information becomes less relevant to predict the missing attribute. For example, the interactions between producers and top five versus not top five composers mixes.

Finally we see Functional Flow performs poorly. This result is not a surprise. Functional Flow iteratively spreads attribute values to nodes with missing values. The average path length is 3.7. One possibility for Functional Flows poor performance is a short average path length and the bipartite nature of the graph causes the algorithm to incorrectly weight values.

Figure 10 shows the precision results for the World Trade data set. In this data set we have three attributes but interpolate only one of them World System Position. World System Position includes 4 categories: core, strong semiperiphery, weak semiperiphery, and periphery; with the following percentages represented 13.75, 21.25, 18.75, and 46.25, respectively. For the export and import numeric values, we divide the values into 5 categories from the minimum to maximum values.

This data set has a very high clustering coefficient: 0.55. Firstly, Functional Flow again performs poorly. Also similar to the Hollywood Film Collaboration data set, the optimal radius is between 2 and 3. However, as we increase the radius, the precision for these methods does not decrease as Figure 9. There is a greater diversity of categories and

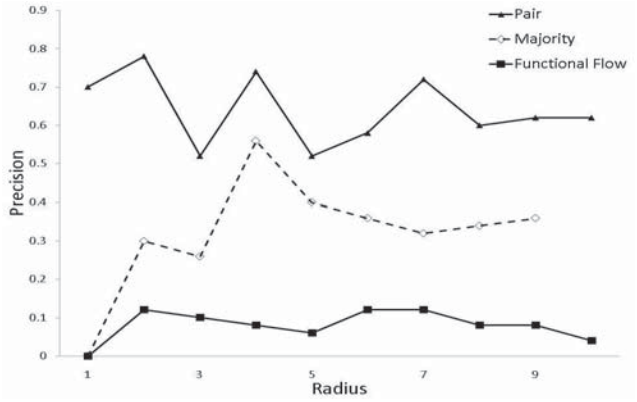

Figure 6.9: Hollywood Film Cooperation Results.

interactions in this data set. Pair has a slightly higher precision rate than the other methods.

Finally, Figure 11 shows the precision results for the Yeast data set. All methods perform weakly here, with precision rates under 0.6. This is not a surprise since protein function prediction is a known hard problem in biology. The optimal radius is around 3 edges away from the query node. Functional Flow performs better than in the other two data sets because now the average path length is greater. Also, since Functional Flow factors distance weight into its predictions, it maintains its precision as the radius increases.

We see that the performance of an interpolation method depends on the data set. An algorithm that can appropriately incorporate graph properties into its method can make better predictions. In all three data sets, Pair performs better than Functional Flow and Majority.

Locality and Setting Radius Parameter

Earlier we presented the user-specified parameter r to set the neighborhood radius. Now we discuss how to set this input. The parameter r determines the induced graph used during interpolation. Also, we examine the effects of varying r to justify a local interpolation approach.

Setting r is tricky. Graphs do not have an intuitive smoothing function to optimize as

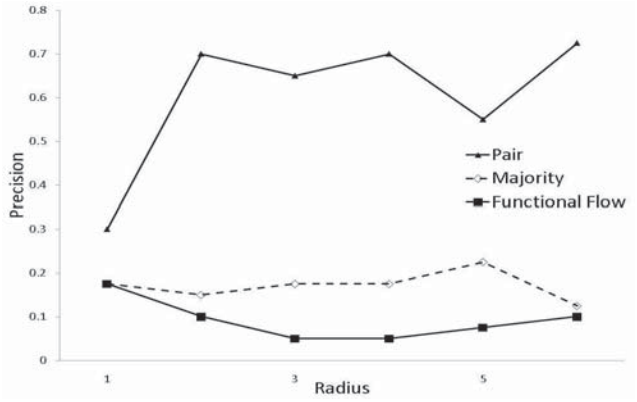

Figure 6.10: World Trade Metal Results.

there exists in geographical interpolation. Ideally, you want a neighborhood with enough relevant information to interpolate the unknown attributes. The size cannot be too large or too small. Regardless of an interpolation method, if the neighborhood is too small, the overall results will be fragmented. Attributes with values will exist only adjacent to other attributes with values. If the neighborhood is too large, the local features will disappear.

For protein networks, the literature uses a radius of 1/4 the diameter of the graph. This value is chosen arbitrarily in most cases. But this value will not work for all graphs. If the graph is strongly connected, fine granularity of local features, and non-uniformly distributed values, there is no proof that choosing r to be 1/4 the diameter is appropriate. The harsh but truthful answer is *r* depends not only on the graph, but also the missing attributes' location in the graph.

Nonetheless, there are some heuristics to use in difficult cases. The first option is to experimentally vary *r* on a prior data set or similar data sets. Then choose the *r* based on precision. Prior data may not always be available. In this case, *r* can be chosen heuristically. The heuristic is to choose the minimum radius which can produce values for the missing value. The intuition behind this heuristic is that a local neighborhood will better reflect the attributes in the node.

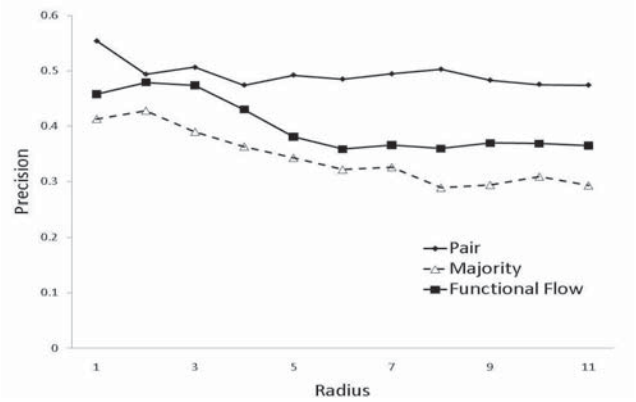

Figure 6.11: Yeast Results.

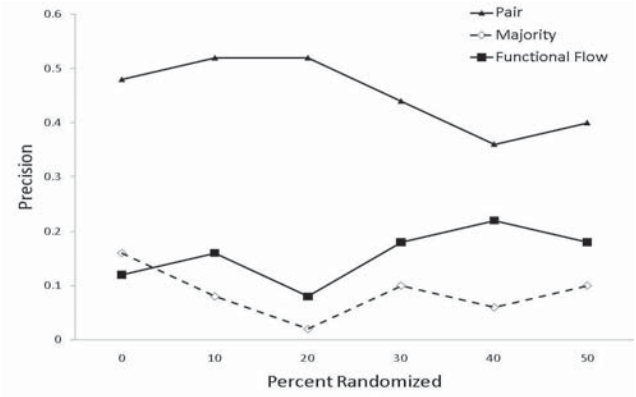

Figure 6.12: Stability Results.

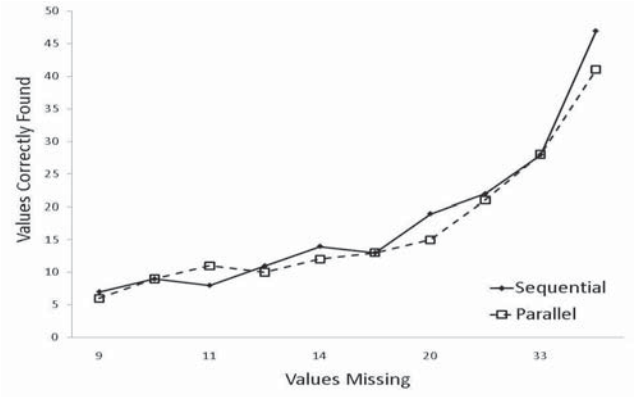

Figure 6.13: Parallel versus Sequential Results.

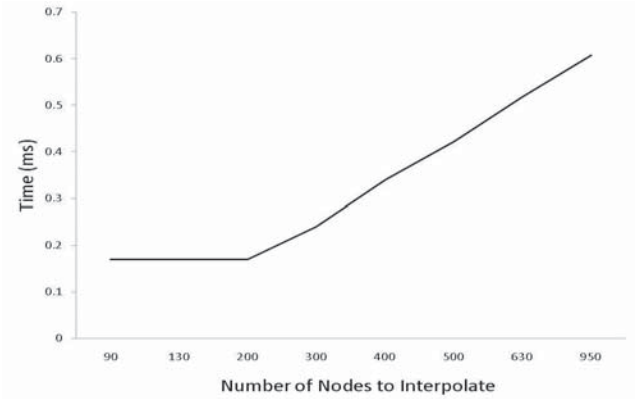

Figure 6.14: Total time to predict grows linearly with the number of missing nodes.

Stability Comparison

Another important property of any graph algorithm relates to how much it changes when the network undergoes small perturbations for example rewiring and value changes. Since we deal with graphs with missing and inaccurate data, we want to know how dependent our method is on the actual graph and up to what changes it can tolerate in interpolating correct values compared to other methods.

For example, a method which relies on shortest path length provides an example of a particularly sensitive measurement - a modification of a single connection may have impact on its value. To test this we perform progressive perturbations on the network and observe the respective relative variations in interpolation effectiveness. The method we use is rewiring. Starting with the original graph, we iteratively choose two random edges to interchange their corresponding attached vertices. We compare to Functional Flow and Majority.

We randomize the graph in increasing 10% intervals, up to 50% of the edges randomized. The randomization process selects two edges (u,v) and (s,t) where $u \neq v \neq s \neq t$ and (u,t), (s,v) are currently not edges. Then (u,v), (s,t) are deleted and (u,t), (s,v) are inserted into the network.

Figure 12 shows the results from a cross validation experiment on the Hollywood Film Collaboration data set using a radius of 2 and returning 1 result. Pair maintains its stability similar to Majority. Also we see Functional Flow slightly improves. This result is due to the new edges creating paths for the network flow.

Parallel versus Sequential

We present both parallel and sequential interpolation methods. Now we compare both methods at increasing numbers of missing values in the World Trade data set. We use a radius of 2 based on the precision results. The sequence is per node and ordered by decreasing node degree. We choose degree as our sequential method so higher degree nodes percolate their interpolated values throughout the graph. Figure 13 shows the number of values correctly predicted by both methods. There is not a big difference between both methods from 0 to 30 values (of the total 80) missing. From 50 to 65 values missing, sequential predicts more correct values. But as the number of missing values increases beyond 65, both methods do poorly, as expected.

Runtime

We present the time results for our parallel Bayesian approach. Although we use the parallel model, we do not make predictions simultaneously. We run the experiment on an Intel Pentium 4 with 1 GB of RAM.

We perform two timing analysis tests using synthetic graphs. We create a set of synthetic scale free Barabasi-Albert model random graphs [4] maintaining a minimum of 2 edges per node. We also create one single-valued attribute per node with 13 possible values. We select a radius of 2 because in our precision experiments this radius often provides the best results.

In the first time experiment we record the time to interpolate a set of nodes for increas-

ing sizes of graphs. Figure 14 shows the results. The time to interpolate a set of nodes is independent of the size of the graph. In other words, interpolating n nodes is independent of the number of vertices, $|V|$, where $n \leq |V|$. Here n is 100. We select the same n nodes for each graph.

Next we maintain the number of vertices in the graph, $|V|$ = 2000, but increase the number of missing attributes. We remove attributes randomly from the graph to interpolate. Figure 15 shows the results. The time to interpolate appears to grow linearly with the number of missing attributes.

6.6 Conclusions

We propose an interaction pair interpolation algorithm for graphs with missing attribute values. We present two approaches of single attribute prediction. Our first method compares the equivalence of a query interaction pair with with neighboring interaction pairs. Based on this comparison, we return values for the missing attribute. The second method uses Bayesian inference analysis. We collect the probabilities of the values occurring in the neighboring interaction pairs. Then we use these probabilities to determine the most likely values for the missing attribute.

Given these two approaches, we show both a parallel and sequential method to perform complete graph interpolation. We verify experimentally that our interaction pair based methods perform better than the state-of-the-art prediction methods. We also show that our methods obtain comparable stability results as the other interpolation methods.

CHAPTER VII

Conclusion and Future Work

7.1 Summary

This thesis explores barriers to visual analytics, including challenges in systematically approaching graph exploration tasks and real data issues such as uncertain queries and missing data. These are problems which are commonly encountered in graph visual analytic tasks, as shown in the VAST 2008 phone analytics thesis overview example. To improve the state of visual analytics, this thesis presents a framework with components to address these challenges.

The visual analytic graph algebra supports scalable and modifiable methods for graph visual analytics. We present several demonstrations and examples of the algebra in use. For example, we implement an algebra plugin for Cytoscape, a widely used graph visualization tool. We also present an application of the algebra to analyze high-throughput biological datasets. We present optimization techniques to accelerate the visual exploration and discovery process for repeat workflows. To attack the problem of open-ended visual exploration of graph, we present a multi-modal graph exploration tool, GreenTrellis. Finally, we give techniques and methods to deal with real world messy data. Specifically, we show techniques to predict missing attributes on nodes and how to execute uncertain queries within the visual analytic graph algebra framework.

7.2 Future Work

In the thesis, we assume an exact graph model. For future work, we would like to extend the algebra to uncertain graph models. Another challenging problem is how to maintain the lineage of the different uncertainty values created in a workflow. Also, in GreenTrellis, we utilize computed attributes based on topology to represent signatures. There are many of intrinsic attributes which describe characteristics in the context of the dataset (e.g. height, color, etc.) We would like to extend signatures to include intrinsic attribute signatures which are also locality meaningful. This step involves identifying ways to quantify discrete attributes and understand how to represent frequencies of various intrinsic attributes at growing distances away from nodes. This future work will allow analysts to study intrinsic and computed attributes simultaneously. Finally, since visual analytics is intended for non-expect analysts to explore their data and make decisions, we would like to further examine the human-factor in our framework. We would like to explore which techniques are most amenable to relating the analytics back to the context of the dataset.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] James Abello, Frank van Ham, and Neeraj Krishnan. Ask-graphview: A large scale graph visualization system. *IEEE Transactions on Visualization and Computer Graphics*, 12:669–676, 2006.

[2] Eytan Adar. Guess: a language and interface for graph exploration. In *CHI '06: SIGCHI conference on Human Factors in computing systems*, pages 791–800, Montreal, Quebec, Canada. ACM.

[3] D Archambault, T Munzner, and D Auber. Grouseflocks: steerable exploration of graph hierarchy space. *IEEE Transactions on Visualization and Computer Graphics*, 2008.

[4] D Auber. *Mathematics and Visualization*. Springer, 2004.

[5] D Auber. Tulip a huge graph visualization framework. *In Graph Drawing Software, Mathematics and Visualization*, 2004.

[6] M. Barouni-Ebrahimi, Ebrahim Bagheri, and Ali A. Ghorbani. A frequency mining-based algorithm for re-ranking web search engine retrievals. In *Canadian Conference on AI*, pages 60–65, 2008.

[7] M. Barouni-Ebrahimi and Ali A. Ghorbani. On query completion in web search engines based on query stream mining. In *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, WI '07, pages 317–320, Washington, DC, USA, 2007. IEEE Computer Society.

[8] V Batagelj and A Mrvar. *Program for large network analysis*.

[9] G Battista, P Eades, R Tamassi, and I Tolli. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1999.

[10] Anastasia Bezerianos, Fanny Chevalier, Pierre Dragicevic, Niklas Elmqvist, and Jean-Daniel Fekete. Graphdice: A system for exploring multivariate social networks. *Computer Graphics Forum (Proc. EuroVis 2010)*, 29(3):863–872, 2010.

[11] B. M. Bolstad, Irizarry R. A., Astrand M., and T. P. Speed. A comparison of normalization methods for high density oligonucleotide array data based on bias and variance. *Bioinformatics*, 19:185–193, 2003.

[12] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos E. Scheidegger, Cláudio T. Silva, and Huy T. Vo. Vistrails: visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, SIGMOD '06, pages 745–747, 2006.

[13] S K Card, J D Mackinlay, and B Shneiderman. *Readings in Information Visualization, Using Vision to Think.* Morgan Kaufmann, 1999.

[14] C Chen. *Visualization beyond the Horizon*. Springer, 2004.

[15] The Cytoscape Collaboration. *Cytoscape Users Manual*, 2006.

[16] The Cytoscape Collaboration. *Cytoscape Users Manual*. Institute for Systems Biology and University of California San Diego, 2006.

[17] Bioconductor Core. An overview of projects in computing for genomic analysis. *Technical Reports*, Nov 2002.

[18] Graham Cormode, Flip Korn, S. Muthukrishnan, and Yihua Wu. On signatures for communication graphs. In *In Proc. 24th IEEE-ICDE*, 2008.

[19] Rhodes DR, Yu J, Shanker K, Deshpande N, and et. al. Oncomine: a cancer microarray database and integrated data-mining platform. *Nature*, 6:1–6, February 2004.

[20] N Elmqvist, T N Do, H Goodell, N Henry, and J D Fekete. Zame: Interactive large-scale graph visualization. *IEEE Pacific Visualization Symposium*, 2008.

[21] National Center for Integrative Biomedical Informatics. *ConceptGen: Gene Set Enrichment Testing and Gene Set Relation Mapping Tool*. University of Michigan Medical School, 2009.

[22] Manuel Freire, Catherine Plaisant, Ben Shneiderman, and Jen Golbeck. Manynets: an interface for multiple network analysis and visualization. In Elizabeth D. Mynatt, Don Schoner, Geraldine Fitzpatrick, Scott E. Hudson, W. Keith Edwards, and Tom Rodden, editors, *Proceedings of the 28th International Conference on Human Factors in ComputingSystems, CHI 2010, Atlanta, Georgia, USA, April 10-15, 2010*, pages 213–222. ACM, 2010.

[23] Lise Getoor and Christopher Diehl. Link mining: A survey. *SigKDD Explorations Special Issue on Link Mining*, 7(2), december 2005.

[24] Lise Getoor, Nir Friedman, Daphne Koller, and Benjamin Taskar. Learning probabilistic models of link structure. *Journal of Machine Learning Research*, 3:679– –707, 2002.

[25] Gd96 graph drawing contest. *http://www.research.att.com/conf/gd96contest.html*. 2006.

[26] Rahul Gupta and Sunita Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB'06*, pages 965–976, 2006.

[27] Wook-Shin Han, Jinsoo Lee, Minh-Duc Pham, and Jeffrey Xu Yu. igraph: a framework for comparisons of disk-based graph indexing techniques. *Proc. VLDB Endow.*, 3(1-2), September 2010.

[28] J Hartigan. *Printer graphics for clustering*. 1975.

[29] Jeffrey Heer. Prefuse: a toolkit for interactive information visualization. In *In CHI 05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430. ACM Press, 2005.

[30] Jeffrey Heer and Michael Bostock. Declarative language design for interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1149–1156, November 2010.

[31] Nathalie Henry, Jean-Daniel Fekete, and Michael J. McGuffin. Nodetrix: a hybrid visualization of social networks. *IEEE Transactions on Visualization and Computer Graphics*, 13:1302–1309, 2007.

[32] Ieee infovis 2011. *http://www.visweek.org/visweek/2011/info*. 2011.

[33] Bohyoung Kim, Bongshin Lee, Susan Knoblach, Eric Hoffman, and Jinwook Seo. Geneshelf: A web-based visual interface for large gene expression time-series data repositories. *IEEE Transctions on Visualization and Computer Graphics*, 15:905–912, 2009.

[34] Heidi Lam. A framework of interaction costs in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1149–1156, November 2008.

[35] Robert S. Laramee and Robert Kosara. Challenges and unsolved problems. In *Human-Centered Visualization Environments*, pages 231–254, 2006.

[36] Kanehisa M. and Goto S. Kegg: Kyoto encyclopedia of genes and genomes. *Nucleic Acids Res*, 28:27–30, 2000.

[37] Kanehisa M., Goto S., M.and Tanabe M. Furumichi, and Hirakawa M. Kegg for representation and analysis of molecular networks involving diseases and drugs. *Nucleic Acids Res*, 38:355–360, 2010.

[38] Kanehisa M., Goto S., M. Hattori, Aoki-Kinoshita, M. Itoh, Katayama T., Araki M., and Hirakawa M. From genomics to chemical genomics: new developments in kegg. *Nucleic Acids Res*, 34:354–357, 2006.

[39] Federica Mandreoli, Riccardo Martoglia, Giorgio Villani, and Wilma Penzo. Flexible query answering on graph-modeled data. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 216–227, New York, NY, USA, 2009. ACM.

[40] Miriah Meyer, Tamara Munzner, and Hanspeter Pfister. Mizbee: A multiscale synteny browser. *IEEE Transctions on Visualization and Computer Graphics*, 15:897–904, 2009.

[41] Matthew Michelson, Sofus A. Macskassy, and Steven N. Minton. Flexible query formulation for federated search. In *Proceedings of the 7th Workshop on Information Integration on the Web (IIWeb)*, Pasadena, CA, 2009.

[42] M E J Newman. *The structure and function of complex networks*. Siam, 2003.

[43] Cydney B. Nielsen, Shaun D. Jackman, Inan Birol, and Steven J.M. Jones. Abyss-explorer: Visualizing genome sequence assemblies. *IEEE Transctions on Visualization and Computer Graphics*, 15:881–888, 2009.

[44] Adam Perer and Ben Shneiderman. Integrating statistics and visualization for exploratory power: From long-term case studies to design guidelines. *IEEE Computer Graphics & Applications*, pages 39–51, 2009.

[45] Varambally S, Yu J, and Laxman B. Integrative genomic and proteomic analysis of prostate cancer reveals signatures of metastatic progression. *Cancer Cell*, 8(5):393–406, November 2005.

[46] Anna Shaverdian, Hao Zhou, George Michailidis, and H. V. Jagadish. Algebraic visual analysis: the catalano phone call data set case study. *KDD Workshop on Visual Analytics and Knowledge*, 2009.

[47] Anna Shaverdian, Hao Zhou, George Michailidis, and H. V. Jagadish. A graph algebra for scalable visual analytics. *IEEE Computer Graphics and Applications*, 2012.

[48] Marc A. Smith, Ben Shneiderman, Natasa Milic-frayling, Eduarda Mendes Rodrigues, Vladimir Barash, Cody Dunne, Tony Capone, Adam Perer, and Eric Gleave. Analyzing (social media) networks with nodexl.

[49] Arun Sreekumar. Metabolomic profiles delineate potential role for sarcosine in prostate cancer progression. *Nature*, 457:910–914, Feb 2009.

[50] J Stasko, C Grg, Z Liu, and K Singhal. Jigsaw: Supporting investigative analysis through interactive visualization. *IEEE Symposuim Visual Analytics Science and Technology*, pages 131–138, October 2007.

[51] K Sugiyama. *Graph Drawing and Applications*. World Scientic Publishing, 2002.

[52] Jim Thomas and Joe Kielman. Information Visualization.

[53] Yuanyuan Tian, Richard C. McEachin, Carlos Santos, David J. States, and Jignesh M. Patel. Saga: a subgraph matching tool for biological graphs. *Bioinformatics*, 23(2), 2007.

[54] Christophe Viau, Michael J. McGuffin, Yves Chiricota, and Igor Jurisica. The flowvizmenu and parallel scatterplot matrix: Hybrid multidimensional visualizations for network exploration. *IEEE Transactions on Visualization and Computer Graphics*, 16:1100–1108, 2010.

[55] Martin Wattenberg. Visual exploration of multivariate graphs. In *In Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 811–819. ACM Press, 2006.

[56] C Weaver. Multidimensional visual analysis using cross-filtered views. *IEEE Symp Visual Analytics Science and Technology*, pages 163–170, October 2008.

[57] Zhen Wen and Michelle. X. Zhou. An optimization-based approach to dynamic data transformation for smart visualization. In *Proceedings of the 13th international conference on Intelligent user interfaces*, IUI '08, pages 70–79, 2008.

[58] Graham J. Wills. Nicheworksinteractive visualization of very large graphs. In *Proceedings of Graph Drawing 97*, pages 403–414. Springer-Verlag, 1997.

[59] Nelson Wong and Sheelagh Carpendale. Supporting interactive graph exploration with edge plucking.

[60] P C Wong, H Foote, G Chin Jr, P Mackey, and K Perrine. Graph signatures for visual analytics. *IEEE Transactions on Visualization and Computer Graphics*, 2006.

[61] P C Wong, H Foote, P Mackey, G Chin Jr, H Sofia, and J Thomas. A dynamic multiscale magnifying tool for exploring large sparse graphs. *Information Visualization*, 2008.

[62] Pak Chung Wong, Stuart J. Rose, George Chin Jr., Deborah A. Frincke, Richard May, Christian Posse, Antonio Sanfilippo, and Jim Thomas. Walking the path: a new journey to explore and discover through visual analytics. *Information Visualization*, 5(4):237–249, 2006.

[63] Xifeng Yan, Philip S. Yu, and Jiawei Han. Substructure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, SIGMOD '05, pages 766–777, New York, NY, USA, 2005. ACM.

[64] Zhiyong Zhang and Olfa Nasraoui. Mining search engine query logs for query recommendation. In *IN PROCEEDINGS OF THE 15TH INTERNATIONAL CONFERENCE ON WORLD WIDE WEB (WWW06*, pages 1039–1040. ACM Press, 2006.

[65] Peixiang Zhao and Jiawei Han. On graph query optimization in large networks. *Proc. VLDB Endow.*, 3(1-2), September 2010.

[66] Hao Zhou, Anna Shaverdian, George Michailidis, and H. V. Jagadish. Multiple step social structure analysis with cytoscape. *Visual Analytics Science and Technology*, 2009.