Tracking Hidden Groups Using Communications

Sudarshan S. Chawathe[1]

Computer Science Department
University of Maryland
College Park, Maryland 20742, USA
chaw@cs.umd.edu

Abstract. We address the problem of tracking a group of agents based on their communications over a network when the network devices used for communication (e.g., phones for telephony, IP addresses for the Internet) change continually. We present a system design and describe our work on its key modules. Our methods are based on detecting frequent patterns in graphs and on visual exploration of large amounts of raw and processed data using a zooming interface.

1 Introduction

Suppose a group of suspicious agents (henceforth, suspects) has been identified based on some a priori knowledge. Instead of taking immediate action to stop the suspicious activities, it is often prudent to carefully monitor the suspects and their communications in order to maximize the detection of suspects (expand the group) and uncover the nexus of activity (locate the key or controlling agents). Unfortunately, the suspects typically do not communicate using easily identifiable sources. For example, a ring of car thieves may continually change phone numbers (using prepaid cellular phones, short-term pager numbers, etc.). Similarly, globally dispersed agents planning a distributed denial-of-service attack on the cyber-infrastructure typically do not use the same IP address for very long. Such behavior makes it very difficult to accurately and efficiently track groups of suspects over extended periods of time. In this paper, we describe a strategy to solve this problem by using a combination of automated and human-directed techniques. We begin by describing the problem more precisely.

Problem Development We will use the term *agents* to denote real-world entities (typically, humans) that we are interested in monitoring. However, these agents are not directly observable and their real-world identities are, in general, unknown. That is, we do not have any method to directly track the actions of the agents. Instead, all we can observe is the communications between such agents. The medium used for such communication may be a phone network, the Internet, physical mail, etc. We refer to it as the *network* in general. We will use the term *nodes* to denote the devices used to communicate using this network (e.g., phone numbers in a telephone network, IP addresses on the Internet). A key feature of nodes is that they are, by virtue of their connections to the network,

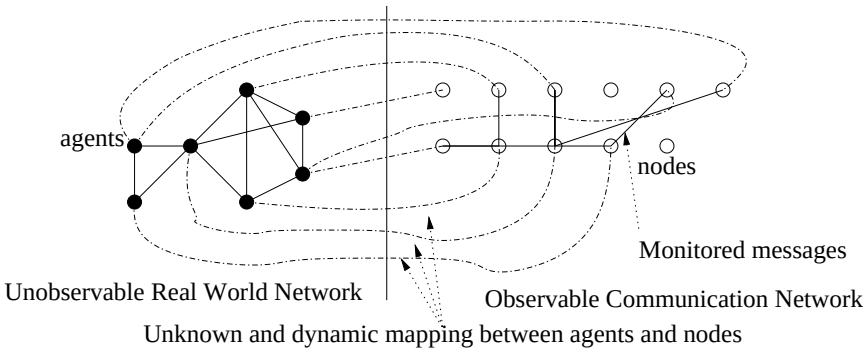


Fig. 1. The tracking problem

easily identifiable and observable. Agents use nodes to communicate on the network. (For example, people use phone numbers to communicate using the phone network, and IP addresses to communicate using the Internet.)

A group of communicating suspects is called a *s-group*. Note that since suspects are, in general, not directly observable, neither are s-groups. At a given point in time, there is a group of nodes (in the communication network) corresponding to the agents in an s-group; we refer to this group of nodes as a *n-group*. In contrast with s-groups, n-groups are easily observable. For example, the group of phone numbers used by a ring of car thieves in the past few days forms a n-group. Over time, the n-group corresponding to a given s-group changes. For example, the ring of thieves is likely to be using a completely different set of phone numbers two months from now. The problem at hand is then the problem of tracking s-groups by observing only the n-groups. By observing a n-group, we mean tracking the communications between the nodes in the group.

In this paper, we assume that the only information we can obtain from the communication network is a timestamped list of inter-node messages. We use the term messages in a general sense. In a phone network, a message is a phone call; on the Internet, a message may be a TCP connection. More precisely, monitoring the network yields a list of tuples of the form (n_1, n_2, t, A) indicating a message from n_1 to n_2 at time t. We use A to denote a list of additional attributes, which depend on the particulars of the communication network and the monitoring methods. In a phone network, A includes attributes such as the length of the call. On the Internet, A includes the source and destination ports associated with a TCP connection and other connection parameters. It is convenient to regard this stream of tuples as the edges of a *connection multigraph* whose nodes represent communication network nodes (e.g., phone numbers) and whose edges represent messages annotated with additional attributes (e.g., phone calls with durations).

In most networks, such a list is never-ending and therefore better modeled as a *stream* of tuples. Another characteristic of the data from network monitoring is that it is typically produced at a very high rate. For example, call records

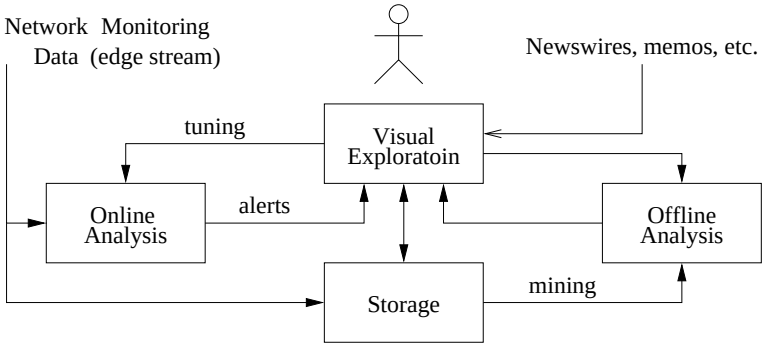


Fig. 2. System architecture

on a phone network and TCP connection build-ups and break-downs occur at a very high rate. It is important to analyze such stream data using *online methods* that detect important patterns as early as possible. (For example, detecting that a ring of thieves is about to move to another state or country may prompt immediate action if it is detected in a timely manner.) Further, indiscriminately storing such stream data can exhaust even the large amounts of inexpensive storage currently available. Storing the data indiscriminately also makes it more difficult to operate on the data as less interesting data is likely to slow access to interesting data. On the other hand, many of the kinds of operations required by this application are not likely to yield to purely online methods. For example, many data mining algorithms require random access to data on disk and cannot be easily modified for the restrictions of stream data. Thus, a practical solution is likely to require both online and offline analysis methods that operate cooperatively.

So far, we have not indicated how the results of the automated or semi-automated methods suggested above are presented to the analyst responsible for decisions, nor have we indicated how such analysts may use their knowledge to direct and guide the tracking process. A simple solution here is to process data in batches, and provide input in batches. For example, a detective may analyze the output of the tracking method from yesterday and adjust the input parameters for guiding the method when it is run on today's data. This solution has problems analogous to those encountered by batch-based solutions to the tracking problem. Again, it is desirable to provide methods that permit online viewing of the results of tracking and immediate fine-tuning of the tracking process. Assuming we have at hand streaming methods for tracking s-groups, we need methods for visualizing, searching, and manipulating the streaming and dynamic data generated by these methods.

System Architecture Figure 2 depicts the high-level architecture of our system for tracking s-groups. The monitoring devices on the network (e.g., instrumented routers on the Internet) produce a stream of tuples, each of which describes a

message between nodes. This stream of tuples is sent to both the online analysis module and the storage module. The storage module is responsible for recording the stream and merging it with the archived data at suitable intervals (say, every 24 hours). The online analysis module uses the stream to trigger detection features based on the archived data and input from the analyst. The offline analysis module is where methods that are not suited to stream processing are implemented. These methods can be classified as data mining or pattern detection methods that require random access to data. The exploration module includes a graphical user interface and, more important, implementation of methods for quickly assimilating vast amounts of data at varying levels of detail. The data includes the stream data processed to varying degrees, the results of the online and offline analysis modules, and an integration with external data sources that are relevant to an analyst's decision making process (e.g., newswire articles, police reports, memos).

In Section 2, we describe methods for detecting frequent patterns in the connection graph. These methods form the building blocks for of the offline analysis module. Section 3 describes methods for exploring large volumes of graph data using a zooming interface that form the basis of the exploration module. We discuss related work briefly in Section 4 and conclude in Section 5. Due to space constraints, we do not discuss the online analysis module here, and refer the interested reader to [5] for details.

2 Detecting Frequent Patterns

In this section, we describe our method for detecting hidden groups by analyzing large volumes of historical connection data obtained by network monitoring. This method is part of the static analysis module of Figure 2. Recall that in this module, we are given a database consisting of a communication graph that forms a historical record of messages between nodes and we wish to detect potential s-groups for further investigation (and to serve as inputs for the online analysis module). The goal is to help an analyst detect s-groups by highlighting patterns in the data. The kinds of patterns of interest to analysts are likely to be varied and complex, and we do not attempt to completely automate the task of detecting them. Instead, our approach is to provide efficient implementation of a few key operations that the analyst may use to investigate the data based on real-world knowledge. In particular, we focus on the efficient implementation of an operation that is not only useful on its own, but also forms the building block for more sophisticated analysis methods (both automated and human directed). This operation is the detection and enumeration of *frequently occurring patterns*, which are informally patterns of communicating nodes occur frequently enough to be of potential interest for a detailed data analysis. (Such frequently occurring patterns are to our problem what *frequent itemsets* are to the problem of mining market basket data [1].)

The main idea behind our method, which is called *SEuS* (Structure Extraction using Summaries) is the following three-phase process: In the first phase

(*summarization*), we preprocess the given dataset to produce a concise summary. This summary is an abstraction of the underlying graph data. Our summary is similar to data guides and other (approximate) typing mechanisms for semistructured data [12, 15, 4]. In the second phase (*candidate generation*), our method interacts with a human analyst to iteratively search for frequent structures and refine the support threshold parameter. Since the search uses only the summary, which typically fits in main memory, it can be performed very rapidly (interactive response times) without any additional disk accesses. Although the results in this phase are approximate (a superset of final results), they are accurate enough to permit uninteresting structures to be conservatively filtered out. When the analyst has filtered potential structures using the approximate results of the search phase, an accurate count of the number of occurrences of each potential structure is produced by the third phase (*counting*).

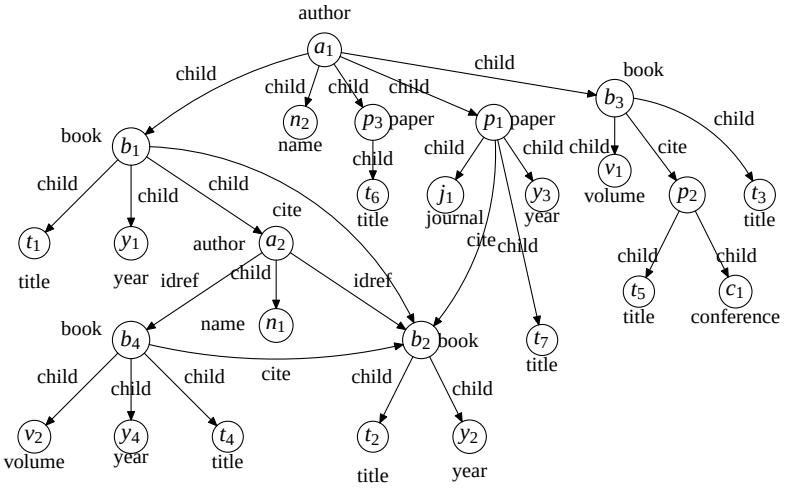


Fig. 3. Example input graph

Users are often willing to sacrifice quality for a faster response. For example, during the preliminary exploration of a dataset, one might prefer to get a quick and approximate insight into the data and base further exploration decisions on this insight. In order to address this need, we introduce an approximate version of our method, called L-SEuS. This method only returns the top-n frequent structures rather than all frequent structures. We present only a brief discussion of SEuS below, and refer the reader to [11] for a detailed discussion both SEuS and L-SEuS.

Summarization We use a data summary to estimate the support of a structure (i.e., the number of subgraphs in the database that are isomorphic to the structure). The summary is a graph with the following characteristics. For each

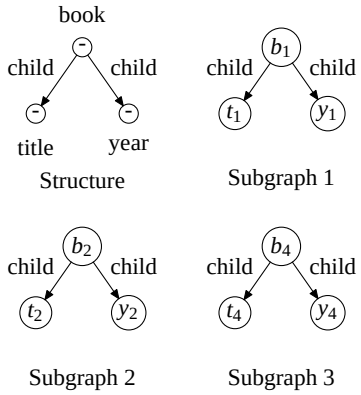


Fig. 4. A structure and its three instances

distinct vertex label l in the original graph G, the summary graph $\mathcal{X}$ has an l-labeled vertex. For each m-labeled edge (v_1, v_2) in the original graph there is an m-labeled edge (l_1, l_2) in $\mathcal{X}$, where l_1 and l_2 are the labels of v_1 and v_2, respectively. The summary $\mathcal{X}$ also associates a counter with each vertex (and edge) indicating the number of vertices (respectively, edges) in the original graph that it represents. For example, Figure 5 depicts the summary generated for the input graph of Figure 3.

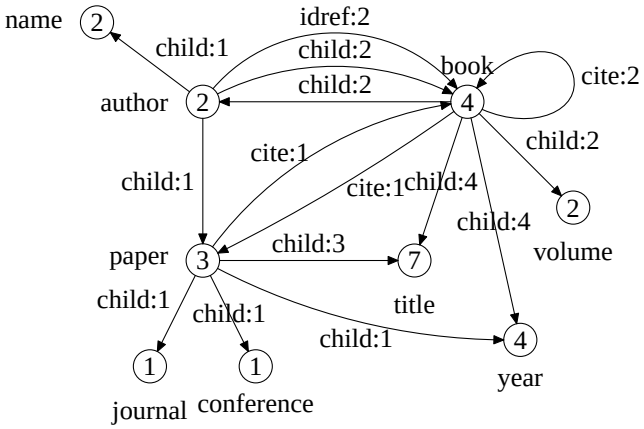


Fig. 5. Summary graph

We use the summary $\mathcal{X}$ to estimate the support of a structure S as follows: By construction, there is at most one subgraph of $\mathcal{X}$ (say, S') that is isomorphic to S. If no such subgraph exists, then the estimated (and actual) support of S is 0. Otherwise, let C be the set of counters on S' (i.e., C consists of counters

on the nodes and edges of S'). The support of S is estimated by the minimum value in C. Given our construction of the summary, this estimate is an upper bound on the true support of S.

Candidate Generation The candidate generation phase is a simple search in the space of structures isomorphic to at least one subgraph of the database. We maintain two lists of structures: *open* and *candidate*. In the open list we store structures that have not been processed yet (and that will be checked later). The algorithm begins by adding all structures that consist of only one vertex and pass the support threshold test to the open list. The rest of the algorithm is a loop that repeats until there are no more structures to consider (i.e., the open list is empty.) In each iteration, we select a structure (S) from the open list and we use it to generate larger structures (called S's *children*) by calling the *expand* subroutine, described below. New child structures that have an estimated support greater than the threshold are added to the open list. The qualifying structures are accumulated in the candidate list, which is returned as the output when the algorithm terminates.

Given a structure S, the *expand* subroutine produces the set of structures generated by adding a single edge to S (termed the children of S). In the following description of the *expand*(S) subroutine, we use $S(v)$ to denote the set of vertices in S that have the same label as vertex v in the data graph and $V(s)$ to denote the set of data vertices that have the same label as a vertex s in S. For each vertex s in S, we create the set $addable(S, s)$ of edges leaving some vertex in $V(s)$. This set is easily determined from the data summary: It is the set of out-edges for the summary vertex representing s. Each edge $e = (s, v, l)$ in $addable(S, s)$ that is not already in S is a candidate for expanding S. If $S(v)$ (the set of vertices with the same label as e's destination vertex) is empty, we add a new vertex x with the same label as v and a new edge (s, x, l) to S. Otherwise, for each $x \in S(v)$ if (s, x, l) in not in S, a new structure is created from S and e by adding the edge (s, x, l) (an edge between vertices already in S). If s does not have an l-labeled edge to any of the vertices in $S(v)$, we also add a new structure which is obtained from S by adding a vertex x' with the same label as v and an edge (s, x', l).

Support Counting Once the analyst is satisfied with the structures discovered in the candidate generation phase, she may be interested in finalizing the frequent structure list and getting the exact support of the structures. This task is performed in the support counting phase.

Let us define the size of a structure to be the number of nodes and edges it contains; we refer to a structure of size k as a *k-structure*. From the method used for generating candidates (Section 2), it follows that for every k-structure S in the candidate list there exists a structure S_p of size $k-1$ or $k-2$ in the candidate list such that S_p is a subgraph of S. We refer to S_p as the *parent* of S in this context. Clearly, every instance I of S has a subgraph I' that is an instance of S_p. Further, I' differs from I only in having one fewer edge and, optionally, one fewer vertex. We use these properties in the support counting process.

Determining the support of a 1-structure (single vertex) consists of simply counting the number of instances of a like-labeled vertex in the database. During the counting phase, we store not only the support of each structure (as it is determined), but also a set of pointers to that structure's instances on disk. To determine the support of a k-structure S for $k > 1$, we revisit the instances of its parent S_p using the saved pointers. For each such instance I', we check whether there is a neighboring edge and, optionally, a node that, when added to I' generates an instance I of S. If so, I is recorded as an instance of S.

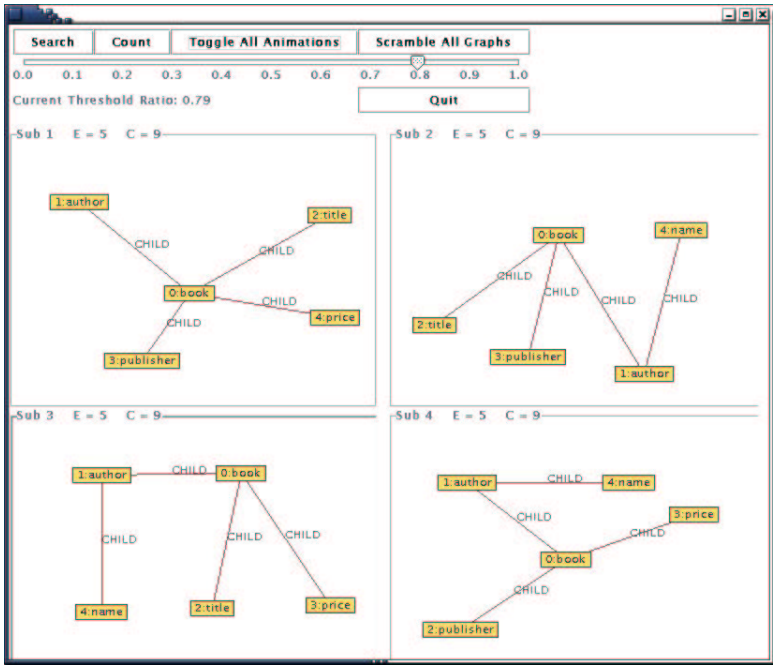


Fig. 6. A screenshot of the SEuS system

3 Visual Exploration

In this section, we describe methods for implementing the interface module of Figure 2. Recall that the task of this module is to help the analyst assimilate the output of the automated analysis modules (offline and online) as well as the external data feed (newswire articles, intelligence reports, etc.). The interconnections between data items from different sources are of particular interest. In this module, we model data as a multiscale graph in which nodes represent data items and edges represent the relationships among them. At a high level, this graph aggregates many data items into one node; at the lowest level, each node

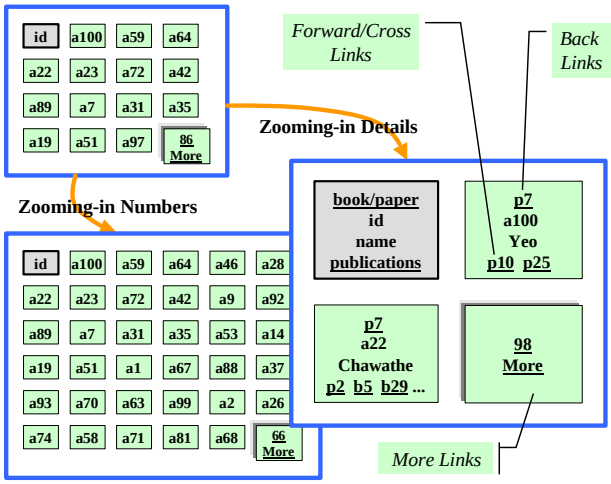


Fig. 7. Two kinds of logical zooming

represents a single data item or concept (e.g., a phone number). This representation allows the analyst to work at a level of abstraction best suited to the task at hand. We have implemented methods for exploring such graphical data at varying levels of detail as part of our *VQBD* system [6], and we describe the key ideas below.

Although VQBD is extensible and incorporates many features for the power user, it is designed to be accessible to a casual user. To this end, the basic modes of interacting with the system are very simple. At all times, the VQBD display consists of a single window with a graphical representation of the XML data. Although, as we shall see below, this representation may be the result of some complex operations, the user interface is always the same: There are nodes (boxes) representing data elements (often summarized) and arcs (lines) representing relationships among them. There are no tool-bars, scroll-bars, sliders, or other widgets. We believe this simplicity is key to usability by a casual user. The basic modes of controlling, described below, VQBD are also simple and unchanging. The first three are meant for the casual user, while the next two are for users who have gained more experience with the system.

Panning The displayed objects can be moved in any direction relative to the canvas by a dragging motion with the left button of the mouse.

Zooming The display may be zoomed in (or out) by a right- (respectively, left-) dragging motion with the right mouse button. VQBD uses the position of the

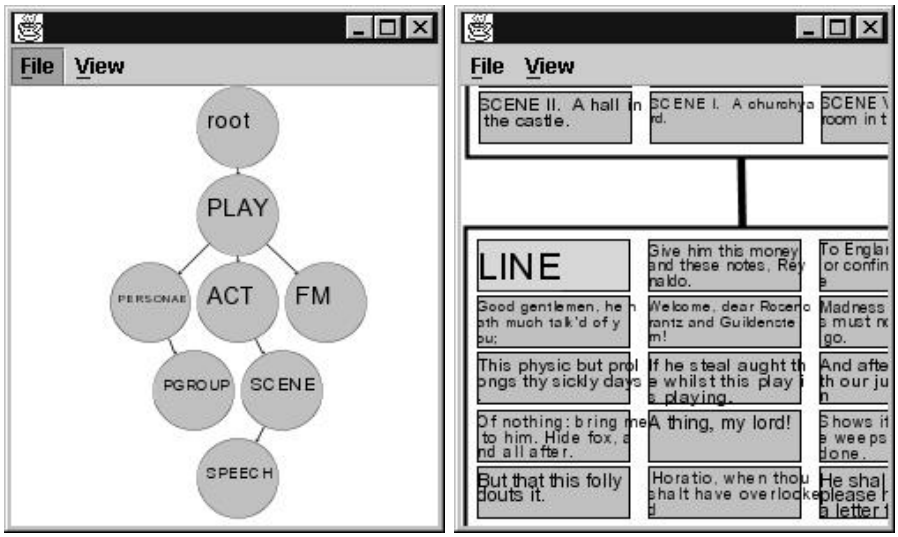
pointer to determine the type of zooming. If the pointer is outside all graphical object then the result is simple graphical zooming (e.g., larger objects, bigger fonts). If the pointer is inside a graphical object then the data resolution of that object, and any others of a similar type, is increased. For example, consider the screenshot in Figure 3. The lower part represents *speech* and *line* objects and includes sample values from the input document. Zooming in with the pointer inside the larger box (representing the *collection* of line objects) results in the display of a larger number of sample speech objects. Zooming in with the pointer inside one of the smaller boxes representing an *individual* line object displays that object in more detail (more text). Figure 7 illustrates these two modes of zooming. In the case of other visualization modules (e.g., histograms), zooming results in actions appropriate to that module (e.g., histogram refinement).

Link Navigation Clicking on a link causes the display to recenter itself around the target of the link at an appropriate zoom level. Following the design method of the *Jazz* toolkit, such link navigation is not instantaneous; instead it occurs at a speed that allows the viewer to discern the relative positions of the referencing and referenced objects. In addition to selecting an appropriate graphical zoom level, VQBD automatically picks a suitable *logical zoom level*. For example, a collection of numbers that is too large to display in its entirety is often presented as a histogram.

View Change While VQBD automatically selects an appropriate method for visualizing data at the available resolution, the user may override this selection a pop-up menu bound to the middle mouse button. For example, a user interested in the highest values in a collection of numbers may force VQBD to change the view from histogram to sorted list.

Querying The XML document may be queried using a query-by-example interface. This interface permits users to specify selection conditions as annotations on displayed objects. In addition, the user may mark objects as *distinguished objects* for use in queries. Intuitively, these objects can be used as the starting points for query-based exploration. VQBD has built-in query modules for regular expressions and XPath. Additional query modules can be easily added using the plug-in interface. More precisely, these objects are logically inserted into a table that can be used in the from clause of OQL-like queries.

Since we do not have access to realistic monitoring data, we illustrate the key features of VQBD using a sample user session based using Jon Bosak's XML rendition of Shakespeare's *A Midsummer Night's Dream*, available at `http://www.ibiblio.org/xml/examples/shakespeare/`. The system parses the data and graphically and presents a summary of its implicit structure with objects representing the play, acts, scenes, and lines. This structural summary is the default view presented by VQBD. A screenshot appears as Figure 3. Note that the screenshots in Figure 8 are based on a rather small VQBD display (approximately 350x350 pixels). While we picked this size primarily to fit the space



(a) Zoomed out—structural summary

(b) Zoomed in—instances

Fig. 8. Two screenshots of VQBD in action

constraints of this report, it also illustrates how VQBD's zooming interface allows it to function effectively at this size. In this example, the summary is small enough to be displayed in its entirety. However, when the summary is larger (or the screen smaller), the panning and graphical zooming features of VQBD are used to view the summary.

Now suppose the analyst zooms in on the speech object using a dragging motion with the right mouse button. Initially, the zooming results in standard graphical results (larger objects, higher resolution text, etc.). However, as soon as the object becomes large enough to display graphical elements within it, the graphical zooming is accompanied by a logical zooming: a few sample elements are displayed. VQBD displays randomly sampled elements, with the number of displayed elements increasing as the available space increases as a result of the zooming in operation. Figure 8(b) is a screenshot at this stage of exploration. In addition to details of the speech and line elements, details of scene elements (appearing above the speech elements in this figure as in Figure 8(b)) are partially visible, providing a useful context. These figures do not convey the colors used by VQBD for indicating many relationships, including grouping elements based on parents (enclosing elements). When a sample element is displayed in this manner, VQBD reads its attributes and sub-elements to pick a short string that distinguishes the element from others with the same tag. This string is

displayed within the object representing the element on screen. In our example, VQBD uses the scene titles to identify scene elements on screen. At this stage, the analyst also has the option of single-clicking on any of the displayed objects, causing VQBD to display all details of the selected object. For example, clicking on the scene object labeled *A hall in the castle* results in displaying the scene in greater detail (as much as will fit in the VQBD window). Note that this clicking action is simply an accelerated form of zooming; the same result could be achieved by zooming in to the scene object. Subelements of the scene element are displayed as active links that can be activated in order to smoothly transport the display to the referenced object. This link-based navigation can be freely interleaved with zooming. Zooming out at this point results in VQBD retracing its steps, displaying data in progressively less detail until we are back at the original structural summary view.

In addition to browsing data in this manner, an analyst may also query data using the VQBD interface. For example, if a scene object is selected as the origin of a search for the string *Lysander*, VQBD executes the query and highlights objects in the query result. In our sample data, the query string matches elements of different types (two *persona* elements, one *stagedir* element, and several *speaker* and *line* elements). If the current resolution is insufficient to display individual objects, only the structural summary objects corresponding to the individual objects are highlighted. To view the query results in detail, one may zoom in as before. Unlike the earlier zooming action, which displayed a random sample of all elements corresponding to the summary object, VQBD now displays a sample chosen only from the elements in the query result. When all elements in the query result have been displayed, further zooming results in a random selection from the remaining elements (as before). (Colors are used to distinguish the elements in the query result from the rest of the elements.) This exploration of query results may be interleaved with zooming, panning, query refinement, and other VQBD operations.

4 Related Work

There is a long history of work on network and graph analysis. However, many of the methods do not scale to the amount of data generated by the network monitoring situations that interest us. For high-volume data, work on *Communities of Interest* [10, 9] is perhaps the closest to our work. A method for managing high-volume call-graph data from a phone network based on daily merging of records is described in [10].

There is work on structure discovery in specific domains; a detailed comparison of several such methods appears in [7]. We are more interested in domain independent methods such as CLIP and Subdue [16, 8]. The method of Section 2 differs from these in its use of a summary structure to yield an interactive system with high throughput. A detailed discussion and performance study appears in [11]. AGM [13] is an algorithm for finding frequent structures that uses an algorithm similar to the apriori algorithm for market basket data [2]. The FSG [14]

is similar to AGM but uses a sparse graph representation that minimizes storage and computation costs. The FREQT algorithm is based on the idea of discovering tree structures using by attaching nodes to only the rightmost branches of trees [3].

The general idea of using a succinct summary of a graph for various purposes has a large body of work associated with it. For example, this idea is developed in semistructured databases as graph schemas, representative objects, and data guides, which are used for constraint enforcement, query optimization, and query-by-example interfaces [4, 15, 12].

5 Conclusion

We described and formalized the problem of tracking hidden groups of entities using only their communications, without a priori knowledge of the communication device identifiers (e.g., phone numbers) used by the entities. We discussed the practical constraints on the environment in which this problem must be solved and presented a system architecture that combines offline analysis, online analysis, and interactive exploration of both raw and processed data. We described our work on methods that form the basis of some of the system modules. We have conducted detailed evaluation of these methods by themselves and are now working on assembling and evaluating the system as a whole.

Acknowledgments

Shayan Ghazizadeh helped design the and implement the SEuS system. Jihwang Yeo and Thomas Baby implemented parts of the VQBD system. This work was supported by National Science Foundation grants in the CAREER (IIS-9984296) and ITR (IIS-0081860) programs.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining associations between sets of items in massive databases. *SIGMOD Record*, 22(2):207–216, June 1993.
2. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th International Conference Very Large Data Bases*, pages 487–499. Morgan Kaufmann, 1994.
3. Tatsuya Asai, Kenji Abe, Shinji Kawasoe, et al. Efficient substructure discovery from large semi-structured data. In *Proc. of the Second SIAM International Conference on Data Mining*, 2002.
4. P. Buneman, S. Davidson, M. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proceedings of the International Conference on Database Theory*, 1997.
5. Sudarshan S. Chawathe. Tracking moving clutches in streaming graphs. Technical Report CS-TR-4376 (UMIACS-TR-2002-56), Computer Science Department, University of Maryland, College Park, Maryland 20742, May 2002.

6. Sudarshan S. Chawathe, Thomas Baby, and Jihwang Yeo. VQBD: Exploring semistructured data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, Santa Barbara, California, May 2001. Demonstration Description.
7. D. Conklin. Structured concept discovery: Theory and methods. Technical Report 94-366, Queen's University, 1994.
8. D. J. Cook and L. B. Holder. Graph-based data mining. *ISTA: Intelligent Systems & their applications*, 15, 2000.
9. Corinna Cortes and Daryl Pregibon. Signature-based methods for data streams. *Data Mining and Knowledge Discovery*, 5:167–182, 2001.
10. Corinna Cortes, Daryl Pregibon, and Chris Volinsky. Communities of interest. In *Fourth International Symposium on Intelligent Data Analysis (IDA 2001)*, Lisbon, Portugal, 2001.
11. Shayan Ghazizadeh and Sudarshan S. Chawathe. SEuS: Structure extraction using summaries. In Steffen Lange, Ken Satoh, and Carl H. Smith, editors, *Proceedings of the 5th International Conference on Discovery Science*, volume 2534 of *Lecture Notes in Computer Science (LNCS)*, pages 71–85, Lubeck, Germany, November 2002. Springer-Verlag.
12. R. Goldman and J. Widom. DataGuides: Enabling query formulation and optimization in semistructured databases. In *Proceedings of the Twenty-third International Conference on Very Large Data Bases*, Athens, Greece, 1997.
13. A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. of the 4th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 13–23, 2000.
14. M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. of the 1st IEEE Conference on Data Mining*, 2001.
15. S. Nestorov, J. Ullman, J. Wiener, and S. Chawathe. Representative objects: Concise representations of semistructured, hierarchial data. In *Proceedings of the International Conference on Data Engineering*, pages 79–90, 1997.
16. K. Yoshida, H. Motoda, and N. Indurkhya. Unifying learning methods by colored digraphs. In *Proc. of the International Workshop on Algorithmic Learning Theory*, volume 744, pages 342–355, 1993.