An Introduction to Temporal Graph Data Management[1]

Udayan Khurana

Computer Science Department
University of Maryland
College Park MD 20740
udayan@cs.umd.edu

Abstract. This paper presents an introduction to the problem of temporal graph data management in the form of a survey of relevant techniques from database management and graph processing. Social network analytics, which focuses on finding interesting facts over static graphs, has gathered much attention lately. However, there hasn't been much work on analysis of temporal or evolving graphs. We believe that efficient techniques to store and query temporal graphs are essential in order to build tools for such analytical tasks. We present previous work done in the areas of temporal relational databases, geospatial databases, graph data management and models of network evolution etc. Also, we present a glimpse of our ongoing work to perform efficient Snapshot Retrieval on Historical Graphs.

Keywords: Temporal Networks, Graph Data Management, Social Network Analysis

1 Introduction

In recent years, there has been a sharp surge in the availability of information network data. Ranging from the digital footprint of online social networks (e.g. Facebook, LinkedIn, Orkut), social media (e.g. Youtube, Flickr, Blogs) to biological networks (e.g. Protein-Protein interaction) and financial transactional networks, large information graphs are ubiquitous. Analysts, sociologists, computer scientists and others are interested in exploring the nature of relationships, patterns, occurrence of communities etc. to understand certain types of behavior or predict events amongst many other objectives. Appropriately, many tools and libraries have been developed to conduct social network analysis (e.g. NodeXL[31], SNAP[32], Blueprints[8]) on

[1] Submitted for the fulfillment of requirement of a Scholarly Paper for Master of Science without thesis option. Supervised by Prof. Amol Deshpande (amol@cs.umd.edu).

graph datasets. However, most of the work in Social Network Analysis till date has focused on examination of static network snapshots. While there has been some recent work in dynamic network visualization [16], demonstrating some interesting scenarios on time evolving networks, we are unaware of any tools that handle dynamic graphs at a reasonable scale. In a recent work, Ahn et al. [2] present an exhaustive taxonomy of temporal visualization tasks. Ren et al. [28] analyze evolution of shortest paths between a pair of vertices over a set of snapshots from the history. Our goal is to explore possibilities to build a graph data management system that can efficiently and scalably support these types of dynamic network analysis tasks over large volumes of data in real-time.

In this paper, we focus on prior work in several connected areas. First, we talk about the work on graph evolution to understand the nature of change in networks. We then talk about related work in temporal relational databases and spatial databases and their applicability in performing basic snapshot retrieval on temporal graphs. We then talk about existing graph data stores and graph query languages. Finally, we talk briefly about our ongoing work in efficient snapshot retrieval on temporal graph datasets.

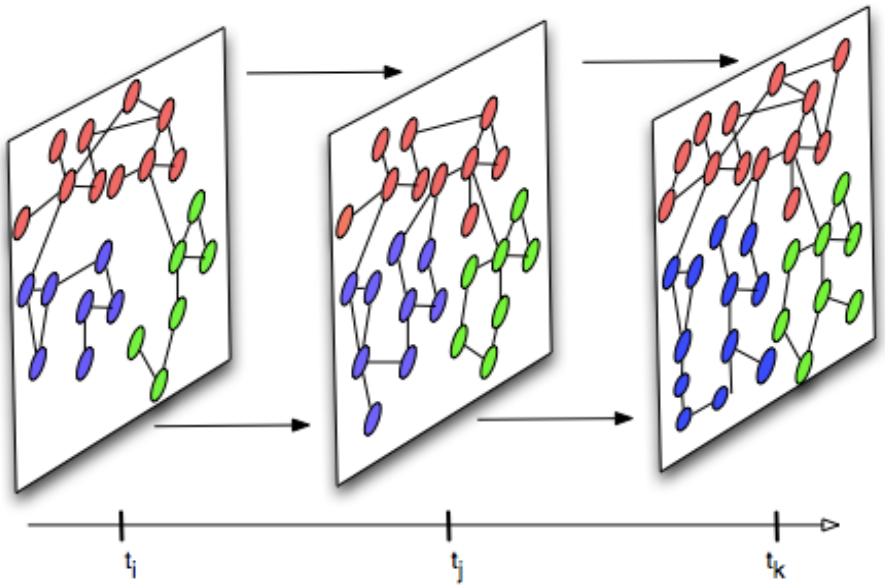

Figure 1. Evolution of communities in a network

2 Evolution of Networks

There has been an increasing interest in dynamic network analysis over the last decade, fueled by the increasing availability of large volumes of temporally annotated network data. Many works have focused on designing analytical models that capture how a network evolves, with a primary focus on social networks and the Web. Barabasi [5] showed that the topology of web follows a growth that can be explained using a power law over degree distribution. Certain nodes in the network act as hubs and there is a heavily tailed degree distribution. This phenomenon, known as "scale-free networks" was recently extended by Leskovec and others [20] with discovery of properties of real networks like decrease in diameter, densification over time and the Forest Fire Model which explains a sharp transition between sparse graphs and graphs that are densifying. Work by Kumar and others [17] tells us that a real network is composed of a) singletons, who do not participate in the network, b) isolated communities, which overwhelmingly display star structure, c) a giant component, anchored by a well-connected core region that persists even in the absence of stars. A complete study of related work in this area is beyond the scope of this work. Overall, the self-similarity and scale-free properties give us an interesting insight into the temporal change in real networks.

There is also much work on understanding how communities evolve, identifying key individuals, and locating hidden groups, in dynamic networks. Berger-Wolf et al. [6, 37], Tang et al. [35] and Greene et al. [13] address the problem of community evolution in dynamic networks. McCulloh and Carley [23] present techniques for social change detection. Asur et al. [4] present a framework for characterizing the complex behavioral patterns of individuals and communities over time.

3 Temporal and Spatial Databases

There is a vast body of literature on temporal relational databases, starting with the early work in the 80's on developing temporal data models and temporal query languages. We won't attempt to present a exhaustive survey of that work, but instead refer the reader to several surveys and books on this topic [9, 33, 26, 36, 11, 33, 29]. The most basic concepts that a relational temporal database is based upon are valid time and transaction time, considered orthogonal to each other. Valid time denotes the time period during which a fact is true with respect to the real world. Transaction time is the time when a fact is stored in the database. A valid-time temporal database permits correction of a previously incorrectly stored fact [33], unlike transaction-time databases where an inquiry into the past may yield the previously known, perhaps incorrect version of a fact.

From a querying perspective, both valid-time and transaction-time databases

can be treated as simply collections of intervals [29], however indexing structures that assume transaction times can often be simpler since they don't need to support arbitrary inserts or deletes into the index. Salzberg and Tsotras [29] present a comprehensive survey of indexing structures for temporal databases. They also present a classification of different queries that one may ask over a temporal database. Under their notation, our focus in this survey is on the valid timeslice query, where the goal is to retrieve all the entities and their attribute values that are valid as of a specific time point.

An optimal solution to answering snapshot retrieval queries is based on an external interval tree, presented by Arge and Vitter [3]. Their proposed index structure uses optimal space on disk, and supports updates in optimal (logarithmic) time. Segment trees [7] can also be used to solve this problem, but may store some intervals in a duplicated manner and hence use more space. Tsotras and Kangelaris [39] present snapshot index, an I/O optimal solution to the problem for transaction-time databases. Salzberg and Tsotras [29] also discuss two extreme approaches to supporting snapshot retrieval queries, called Copy and Log approaches. In the Copy approach, a snapshot of the database is stored at each transaction state, the primary benefit being fast retrieval times; however the space requirements make this approach infeasible in practice. The other extreme approach is the Log approach, where only and all the changes are recorded to the database, annotated by time. While this approach is space-optimal and supports O(1)-time updates (for transaction-time databases), answering a query may require scanning the entire list of changes and takes prohibitive amount of time. A mix of those two approaches, called Copy+Log, where a subset of the snapshots is explicitly stored, is often a better idea.

While these approaches serve can be used for the problem of snapshot retrieval in temporal data, they are insufficient and inflexible for an efficient and general-purpose solution to temporal graph data stores. First, they do not efficiently support multipoint queries that are expected to be very commonly used in evolutionary analysis and need to be optimized by avoiding duplicate reads and repeated processing of the events. Second, to cater to the needs of a variety of different applications, such an index structure needs to be highly tunable, and to allow trading off different resources and user requirements (including memory, disk usage, and query latencies). Ideally one would also like to control the distribution of average snapshot retrieval times over the history, i.e., to be able to reduce the retrieval times for more recent snapshots at the expense of increasing it for the older snapshots (while keeping the utilization of the other resources the same), or vice-versa. For achieving low latencies, the chosen index structure should support flexible pre-fetching of portions of the index into memory and should avoid processing any events that are not needed by the query (e.g., if only the network structure is needed, then we should not have to process any events pertaining to the node or edge attributes). Finally, we would like the index structure to be able to support different persistent storage options, ranging from a hard

disk to the cloud; most of the previously proposed index structures are optimized primarily for disks.

4 Graph Data Management

There has been resurgence of interest in general-purpose graph data management systems in both academia and industry. Several commercial and open-source graph management systems are being actively developed (e.g., Neo4j[2], GBase[3], Pregel [16]). Blueprints [8] is a set of interfaces which links graph stores like Neo4j with graph algorithm APIs like JUNG (Java Universal Network/Graph Framework)[4] to operate upon the underlying graph data. There is much ongoing work on efficient techniques for answering various types of queries over graphs and on building indexing structures for the same. However, there do not exist any graph data management system that focuses on optimizing snapshot retrieval queries over historical graph traces, and on supporting rich temporal analysis of large networks.

There is also prior work on temporal RDF data and temporal XML Data. Motik [24] presents a logic-based approach to representing valid time in RDF and OWL. Several works (e.g., [27, 38]) have considered the problems of subgraph pattern matching or SPARQL query evaluation over temporally annotated RDF data. There is also much work on version management in XML data stores. Most scientific datasets are semistructured in nature and can be effectively represented in XML format [10]. Lam and Wong [19] use complete deltas, which can be traversed in either direction of time for efficient retrieval. Other systems store the current version as a snapshot and the historical versions as deltas from the current version [22]. For such a system, the deltas only need to be unidirectional. Ghandeharizadeh et al. [12] provide a formalism on deltas, which includes a delta arithmetic. All these approaches assume unique node identifiers to merge deltas with deltas or snapshots. Buneman et al. [10] propose merging all the versions of the database into one single hierarchical data structure for efficient retrieval. However, none of that prior work focuses on snapshot retrieval in general graph databases, or proposes techniques that can flexibly exploit the memory-resident information.

[2] http://www.neo4j.org
[3] http://www.graphbase.net
[4] http://jung.sourceforge.net

5 Introduction to HGDB

Historical Graph Database (HGDB) is the part of ongoing work at University of Maryland that is aimed at creating a database optimized for retrieving Historical Snapshots and performing analytical tasks on temporal networks [15]. The core ideas behind this system are to store the historical trace of the network on disk in a space efficient manner, and to load the required graphs on-demand in memory in a compact, non-redundant manner.

Figure 2 shows a high level overview of our system and its key components. At a high level, there are multiple ways that a user or an application may interact with a historical graph database. Given the wide variety of network analysis or visualization tasks that are commonly executed against an information network, we expect a large fraction of these interactions will be through a programmatic API where the user or the application programmer writes her own code to operate on the graph. Such interactions result in what we call snapshot queries being executed against the database system. Executing such queries is the primary focus of this system. In ongoing work, we are also working on developing a high-level declarative query language (similar to TSQL [24]) and query processing techniques to execute such queries against our database. As a concrete example, an analyst who may have designed a new network evolution model and wants to see how it fits the observed data, may want to retrieve a set of historical snapshots and process them using the programmatic API. On the other hand, a declarative query language may better fit the needs of a user interested in searching for a temporal pattern (e.g., find nodes that had the fastest growth in the number of neighbors since joining the network).

The most basic model of a graph over a period of time is as a collection of graph snapshots, one corresponding to each time instance (we assume discrete time). Each such graph snapshot contains a set of nodes and a set of edges. The nodes and edges are assigned unique ids at the time of their creation, which are not reassigned after deletion of the components (a deletion followed by a reinsertion results in assignment of a new id). A node or an edge may be associated with a list of attribute-value pairs; the list of attribute names is not fixed a priori and new attributes may be added at any time. Additionally an edge contains the information about whether it is a directed edge or an undirected edge.

We define an *event* as the record of an atomic activity in the network. An event could pertain to either the creation or deletion of an edge or node, or change in an attribute value of a node or an edge. Alternatively, an event can express the occurrence of a transient edge or node, which is valid only for that time instance instead of an interval (e.g., a “message” from a node to another node). Being atomic refers to the fact that the activity cannot be logically broken down further into smaller events. Hence, an event always corresponds to a single timepoint. So, the valid time interval of an edge, $[t_s,t_e]$, is expressed by two different events, edge addition and

deletion events at t_s and t_e respectively. All events are recorded in the direction of evolving time, i.e., going ahead in time. A list of chronologically organized events is called an *eventlist*.

There are two key data structure components of our system.

1. ***GraphPool*** is an in-memory data structure that can store multiple graphs together in a compact way by overlaying the graphs on top of each other. At any time, the GraphPool contains: (1) the current graph that reflects the current state of the network, (2) the historical snapshots, retrieved from the past using the commands above and possibly modified by an application program, and (3) materialized graphs, which are graphs that correspond interior or leaf nodes in the DeltaGraph, but may not correspond to any valid graph snapshot. GraphPool exploits redundancy amongst the different graph snapshots that need to be retrieved, and considerably reduces the memory requirements for historical queries. More specifically, memory footprint of the system is given by: $|G_c + G_1 + \ldots + G_n| \approx |G_c \cup G_1 \cup G_2 \ldots \cup G_n| + z$, where G_c is the current graph, $G_1 \ldots G_n$ are retrieved snapshots, and z is the small extra overhead of maintaining the overlaid structure.

2. ***DeltaGraph*** is a disk-resident index structure that stores the historical network data using a hierarchical index structure over deltas and leaf-level eventlists (called leaf-eventlists). To execute a snapshot retrieval query, a set of appropriate deltas and leaf-eventlists are fetched and the resulting graph snapshot is overlaid on the existing set of graphs in the GraphPool. The structure of the DeltaGraph itself, called DeltaGraph skeleton, is maintained as a weighted graph in memory (it contains statistics about the deltas and eventlists, but not the actual data). The skeleton is used during query planning to choose the optimal set of deltas and eventlists for a given query.

The data structures are managed and maintained by several system components. HistoryManager deals with the construction of the DeltaGraph, plans how to execute a singlepoint or multipoint snapshot query, and reads the required deltas and eventlists from the disk. GraphManager is responsible for managing the GraphPool data structure, including the overlaying of deltas and eventlists, bit assignment, and post-query clean up. Finally, the QueryManager manages the interface with the user or the application program, and extracts a snapshot query to be executed against the DeltaGraph.

Using DeltaGraph index, snapshot retrieval involves finding the correct and optimal set of deltas and events (through eventlists) to be read from the disk and be loaded into the GraphPool (in memory). Once, the required snapshot or the graph is in GraphPool, it may be used for the desired computational objective, e.g. executing PageRank or determining shortest paths.

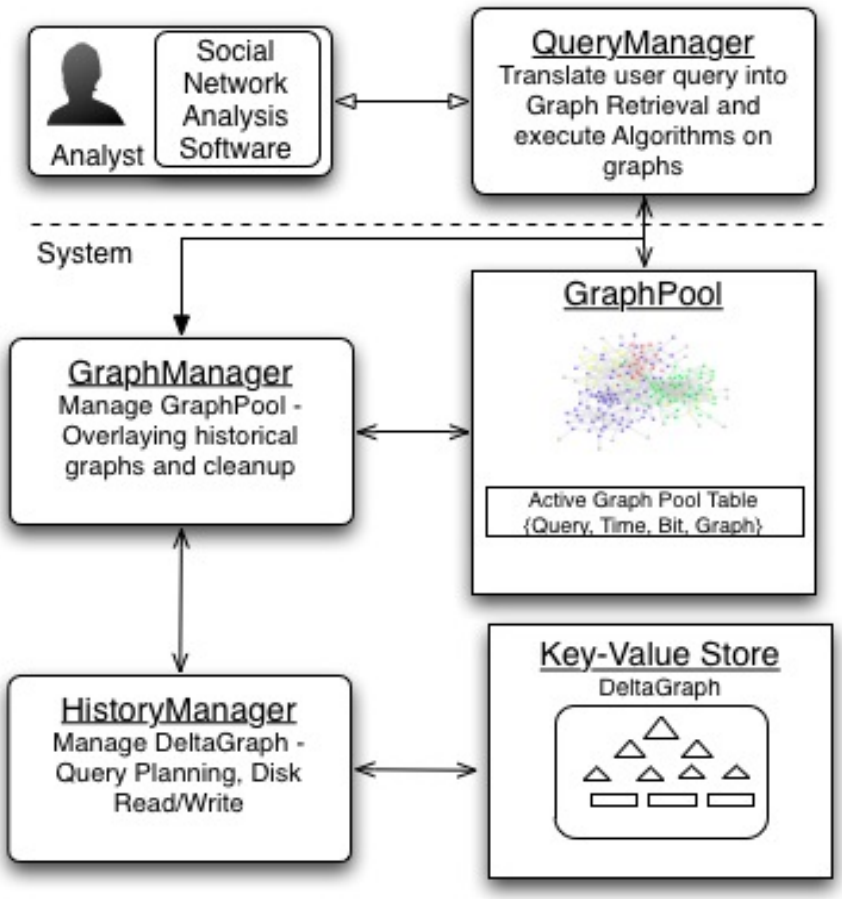

Figure 2. System Architecture for our proposed Historical Graph Data Management System

6 Conclusion

In this paper, we discussed the problem of managing historical or temporal information of large networks. While there has been considerable work in temporal relational data management and graph data management respectively, the area of temporal graph management still lies unaddressed to a large extent. It is clear that temporal social network analytics on any scale of a reasonable size would require an

underlying system for efficiently managing such data. We also presented an outline of our ongoing work on snapshot retrieval on historical graphs.

References

1. C. C. Aggarwal and H. Wang. Graph data management and mining: A survey of algorithms and applications. In Managing and Mining Graph Data, pages 13–68. 2010.
2. J. Ahn, C. Plaisant, and B. Shneiderman. A task taxonomy of network evolution analysis. HCIL Technical Reports, 2011.
3. L. Arge and J. Vitter. Optimal dynamic interval management in external memory. In FOCS, 1996.
4. S. Asur, S. Parthasarathy, and D. Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. ACM TKDD, 2009.
5. A. Barabási. Emergence of scaling in random networks. Science 1999 (286), 509–512.
6. T. Berger-Wolf and J. Saia. A framework for analysis of dynamic social networks. In KDD, 2006.
7. G. Blankenagel and R. Guting. External segment trees. Algorithmica, 12(6): 498–532, 1994.
8. Blueprints, https://github.com/tinkerpop/blueprints/wiki/
9. A. Bolour, T. L. Anderson, L. J. Dekeyser, H. K. T. Wong. The role of time in information processing: a survey. SIGMOD Rec., 1982.
10. P. Buneman, S. Khanna, K. Tajima, and W. Tan. Archiving scientific data. ACM TODS, 29(1):2–42, 2004.
11. C. Date, H. Darwen, and N. Lorentzos. Temporal data and the relational model. Elsevier, 2002.
12. S. Ghandeharizadeh, R. Hull, and D. Jacobs. Heraclitus: elevating deltas to be first-class citizens in a database programming language. ACM TODS, 21(3), 1996.
13. D. Greene, D. Doyle, and P. Cunningham. Tracking the evolution of communities in dynamic social networks. In ASONAM, 2010.
14. H. He and A. Singh. Graphs-at-a-time: query language and access methods for graph databases. In SIGMOD, 2008.
15. U. Khurana, A. Deshpande. Historical Graph Data Management. Submitted to VLDB 2012.
16. U. Khurana, V Nguyen, H. Cheng, Ahn, X. Chen, B. Shneiderman. Visual Analysis of Temporal Trends in Social Networks Using Edge Color Coding and Metric Timelines. IEEE Social Computing 2011.

17. R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In KDD, 2006.
18. A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. SIGOPS Oper. Syst. Rev., 2010.
19. N. Lam and R. Wong. A fast index for XML document version management. In APWeb, 2003.
20. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. ACM TKDD, 2007.
21. G. Malewicz, M. Austern, A. Bik, J. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In PODC, 2009.
22. A. Marian, S. Abiteboul, G. Cobena, and L. Mignet. Change-centric management of versions in an XML warehouse. In VLDB, 2001.
23. I. McCulloh and K. Carley. Social network change detection. Center for the Computational Analysis, 2008.
24. B. Motik. Representing and querying validity time in RDF and OWL: A logic-based approach. In ISWC, 2010.
25. S. Navlakha, C. Kingsford. Network archaeology: Uncovering ancient networks from present-day interactions. PLoS Comput Biol, 2011.
26. G. Ozsoyoglu and R.T. Snodgrass. Temporal and real-time databases: a survey. IEEE TKDE, 7(4):513 –532, aug 1995.
27. A. Pugliese, O. Udrea, and V. Subrahmanian. Scaling RDF with time. In WWW, 2008.
28. C. Ren, E. Lo, B. Kao, X. Zhu, and R. Cheng. On querying historial evolving graph sequences. In VLDB, 2011.
29. B. Salzberg and V. Tsotras. Comparison of access methods for time-evolving data. ACM Comput. Surv., 31(2), 1999.
30. A. Seering, P. Cudre-Mauroux, S. Madden, and M. Stonebraker. Efficient versioning for scientific array databases. In ICDE, 2012.
31. M. Smith, N. Milic-Frayling, B. Shneiderman, E. Rodrigues, J. Leskovec, C. Dunne. NodeXL: a free and open network overview, discovery and exploration add-in for Excel 2007/2010, http://nodexl.codeplex.com/, 2010.
32. SNAP: Stanford Network Analysis Project, http://snap.stanford.edu/
33. R. Snodgrass and I. Ahn. A taxonomy of time in databases. In SIGMOD, pages 236–246, 1985.
34. Richard T. Snodgrass, editor. The TSQL2 Temporal Query Language. Kluwer, 1995.
35. L. Tang, H. Liu, J. Zhang, and Z. Nazeri. Community evolution in dynamic multi-mode networks. In KDD, 2008.
36. A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass (editors). Temporal Databases: Theory, Design, and Implementation. 1993.

37. C. Tantipathananandh, T. Berger-Wolf, D. Kempe. A framework for community identification in dynamic social networks. In KDD, 2007.
38. J. Tappolet and A. Bernstein. Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL. In ESWC, 2009.
39. V. Tsotras and N. Kangelaris. The snapshot index: an I/O-optimal access method for timeslice queries. Inf. Syst., 1995.