Noname manuscript No.
(will be inserted by the editor)

Historical Graph Data Management

Udayan Khurana and Amol Deshpande

the date of receipt and acceptance should be inserted later

Abstract Real world graphs evolve over time, with continuous addition and removal of vertices and edges, as well as frequent change in their attributes. Some examples of such graphs are – phone-call graphs generated by telecommunication service providers, message graphs from social networking sites, and mention-activity graphs formed by Twitter users mentioning one another in their tweets, and so on. For decades, the work in graph analytics was restricted to a static perspective of the graph. Analysis such as finding important (central) entities in a network, groups (clusters) of similar entities, observing graph density, diameter and several other attributes have been well studied in the context of static graph snapshots. In recent years, however, we have witnessed an increasing abundance of timestamped observational data describing various types of temporal information networks, including social networks, biological networks, citation networks, financial transaction networks, communication networks, to name a few. This has fueled an interest in performing richer analysis of graphs, along a temporal dimension. Analysis of history of a graph presents fascinating insights into the underlying phenomena that produced the graph. However, the traditional network data management systems provide inadequate support for such analyses. We present a summary of recent advances in the field of historical graph data management. They involve, compact storage of large graph histories, efficient retrieval of temporal subgraphs, and effective interfaces for expressing historical graph queries are essential for enabling temporal graph analytics.

1 Overview

In recent years, several works have designed analytical models that capture network evolution, with a focus on social networks and the Web [LKF07, KNT10]. Studies analyze how communities evolve [TLZN08], identifying key individuals, and locating hidden groups in dynamic networks [TBWK07], also characterizing the complex behavioral patterns of individuals and communities over time [APU09]. Biologists are interested in discovering historical events leading to a known state of a biological network (e.g., [NK11]). Change in page rank [BCG10], change in centrality of vertices, path lengths of vertex pairs, etc. [PS11], shortest paths evolution [RLK+11], and such, provide useful analytical algorithms for changing graphs. Historical or temporal analyses on graphs span a variety of tasks that vary in the analytical quantity of interest, as well as the scope of the graph locality and time-duration being analyzed. Figure 1 provides a classification on those lines and lists a few examples of graph retrieval and analysis tasks. A more exhaustive taxonomy of temporal tasks is provided by Ahn et al. [APS14].

Typical graph data management systems that are based on static graphs, do not provide sufficient support for the analytical tasks described above. This is due to a fundamental lack in the modeling of the *change* of information. If performed using the conventional graph systems, such tasks become too expensive in storage costs, memory requirements, or execution time, and are often unfriendly or infeasible for an analyst to even express in the first place. The frequent change of a temporal graph also poses a significant challenge to algorithm design, because the overwhelming majority of graph algorithms assume static graph structures. One would have to design special algorithms for each application to accommodate the dynamic

Udayan Khurana
IBM Research AI, TJ Watson Research Center, Yorktown Heights New York NY, USA
E-mail: ukhurana@us.ibm.com

Amol Deshpande
Computer Science Department, University of Maryland, College Park MD, USA
E-mail: amol@cs.umd.edu

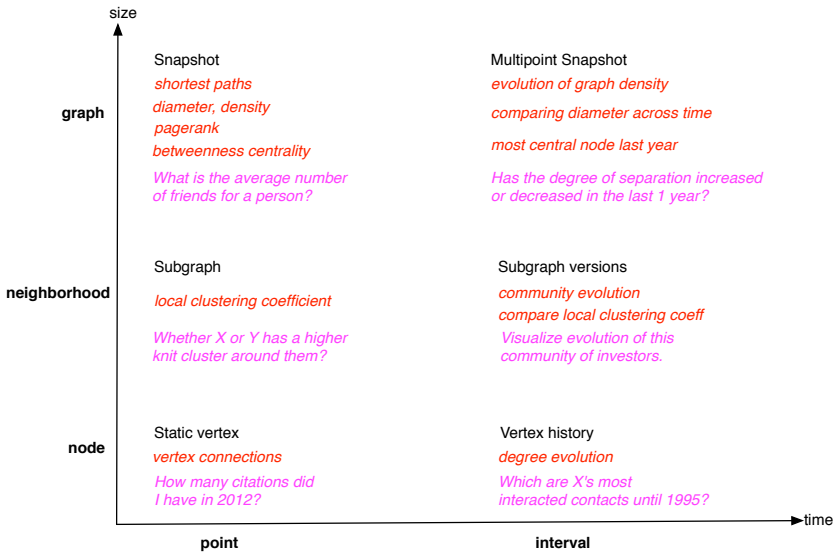

Fig. 1 A temporal graph can be represented across two different dimensions - time and entity. It lists retrieval tasks (black), graph operations (red), example queries (magenta) at different granularities of time and entity size [Khu15].

aspects of graphs. To support general-purpose computations, most of the emerging temporal graph systems adopt a strategy to separate graph updates from graph computation. More specifically, although updates are continually applied to a temporal graph, graph computation is only performed on a sequence of successive *static* views of the temporal graph. For simplicity, most systems adopt a *discretized-time* approach, so that time domain is set of natural numbers, i.e., $t \in \mathcal{N}$. As per the terminology of temporal relational databases, this discussion considers *valid time* (against *transaction time*) as the underlying temporal dimension for historical analyses. 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. It is worth noting that there is a related but orthogonal body of work which we do not touch in this chapter. It deals with the need to do *real-time analytics on the streaming data* as it is being generated; here the scope of the analysis typically only includes the latest snapshot or the snapshots from a recent window. The key challenge there is to be able to deal with the high rate at which the data is often generated.

Broadly speaking, this chapter's focus is to briefly introduce the reader to the recent advances in historical graph data management for temporal graph analytics. There are many different types of analyses that may be of interest. For example, an analyst may wish to study the evolution of well-studied static graph properties such as centrality measures, density, conductance, etc., over time; or, the search and discovery of temporal patterns, where the events that constitute the pattern are spread out over time. Comparative analysis, such as juxtaposition of a statistic over time, or perhaps, computing aggregates such as *max* or *mean* over time, gives another style of knowledge discovery into temporal graphs. Most of all, even a primitive notion of simply being able to access past states of the graphs and performing simple static graph analytics, empowers a data scientist with the capacity to perform analysis in arbitrary and unconventional patterns. Supporting such a diverse set of temporal analytics and querying over large volumes of historical graph data requires addressing several data management challenges. Specifically, we need techniques for storing the historical information in a compact manner, while allowing a user to retrieve graph snapshots as of any time point in the past, or the evolution history of a specific node or a specific neighborhood. Further the data must be stored and queried in a distributed fashion to handle the increasing scale of the data. Finally, there is a need for an expressive, high-level, easy-to-use programming framework that will allow users to specify complex temporal graph analysis tasks. in a data-parallel fashion across a cluster.

2 Storage and Retrieval

Storage of large temporal graphs is challenging and requires careful design. An effective storage system for temporal graphs has a twofold objective. First, the storage should be compact such that the invariant information across multiple graph versions is not stored multiple times. Second, it must allow for efficient retrieval of graph primitives such as snapshot(s), temporal range queries of nodes or neighborhoods, amongst others, as reflected in Figure 1. A straightforward use of static graph datastores can lead to an explosion in storage requirements, and/or incur high fetch latency times. Consider two basic and extreme

approaches from conventional storage methods to support snapshot retrieval queries, referred to as the *Copy* and *Log* approaches, respectively [ST99]. While the Copy approach relies on storing new copies of a snapshot upon every point of change, the log approach relies on storing everything through changes. Their hybrid is often referred to as the *Copy+Log* approach. Copy approach is clearly infeasible for frequently changing graphs because of intractable storage needs; the Log approach, while storing minimal information, makes it hard work to reconstruct any snapshot or temporal subset of the graph.

2.1 Delta-based Encodings

The use of "*deltas*", or graph differences, is a powerful approach to register the changes in a graph over a period of time. If carefully organized, deltas can provide efficient retrieval at low storage costs. The *DeltaGraph* index [KD13], for instance, provides highly efficient retrieval of individual snapshots of the historical graph for specific time instances. It organizes the historical graph data in a hierarchical data structure, whose lowest level corresponds to the snapshots of the network over time, and whose interior nodes correspond to graphs constructed by "combining" the lower level snapshots in some fashion; the interior nodes are typically not valid snapshots as of any specific time point. Neither the lowest-level graph snapshots nor the graphs corresponding to the interior nodes are actually stored explicitly. Instead, for each edge, a *delta*, i.e., the difference between the two graphs corresponding to its endpoints, is computed, and these deltas are explicitly stored. In addition, the graph corresponding to the root is explicitly stored. Given those, any specific snapshot can be constructed by traversing any path from the root to the node corresponding to the snapshot in the index, and by appropriately combining the information present in the deltas. Use of different "combining" functions leads to a different point in the performance-storage trade-off, with *intersection* being the most natural such function. This index structure is especially effective with *multi-snapshot retrieval* queries, which are expected to be common in temporal analysis, as it can share the computation and retrieval of deltas across the multiple snapshots. While it is efficient at snapshot retrieval, is not suitable for fetching graph primitives such as histories of nodes or neighborhoods over specified periods of time. However, *Temporal Graph Index (TGI)* [KD16], an extension of DeltaGraph which partitions deltas in so-called *micro-deltas*, and uses chaining of related microdeltas across horizontal nodes in the DeltaGraph, allows retrieval of different temporal graph primitives including neighborhood versions, node histories, and graph snapshots. Such a micro-delta based design is great for distributed storage and parallel retrieval on a cloud. This allows efficient retrieval of not only entire snapshots, but also of individual neighborhoods or temporal histories of individual neighborhoods.

2.2 Storing by Time Locality

Another powerful approach to store and process temporal graphs is based on time-locality. *Chronos* [HML+14] targets time-range graph analytics, requiring computation on the sequence of static snapshots of a temporal graph within a time range. An example is, the analysis of change in each vertex's PageRank for a given time range. Obviously, the most straightforward approach of applying computation on each snapshot separately is too expensive. Chronos achieves efficiency by exploiting locality of temporal graphs. It also leverages the time locality to store temporal graphs on disk in a compact way. The layout is organized in snapshot groups. A snapshot group G_{t_1,t_2} contains the state of G in the time range $[t_1, t_2]$, by including a checkpoint of the snapshot of G at t_1 followed by all the updates made till t_2. The snapshot group is physically stored as edge files and vertex files in time-locality fashion. For example, an edge file begins with an index to each vertex in the snapshot group, followed by segments of vertex data. The segment of a vertex, in turn, first contains a set of edges associated with the vertex at the start time of the snapshot group, followed by all the edge updates to the vertex. A link structure is further introduced to link edge updates related to the same vertex/edge, so that the state of a vertex/edge at a given time t can be efficiently constructed.

2.3 Indexing using Multiversion Arrays

LLAMA [MMMS15] presents an approach where an evolving graph is modeled as a time series of graph snapshots, where each batch of incremental updates produces a new graph snapshot. The graph storage is read-optimized, while the update buffer is write-optimized. It augments the compact read-only CSR representation to support mutability and persistence. Specifically, a graph is represented by a single vertex table, and multiple edge tables, one per snapshot. The vertex table is organized as a large multi-versioned array (LLAMA) that uses a software copy-on-write technique for snapshotting, and the record of each vertex v in the vertex table maintains the necessary information to track v's adjacency list from the edge

tables across snapshots. The array of records is partitioned into equal-sized data pages, and an indirection array is constructed that contains pointers to the data pages. The indirection array fits in L3 cache. To create a new snapshot, the indirection array is copied, with those references to out-dated pages replaced by those to the newly modified pages. Thus, we do not need to copy unmodified pages across snapshots. LAMA stores 16 consecutive snapshots of the vertex table in each file, so that disk space can be easily reclaimed from deleted snapshots. The edge table for a snapshot i is organized as a fixed-length array that stores adjacency list fragments consecutively, where each adjacency list fragment contains the edges of a vertex added in snapshot i. An adjacency list fragment of vertex v also stores a continuation record, which points to the next fragment for v, or *null* if there are no more edges. To support edge deletion, each edge table maintains a deletion vector, which is an array that encodes in which snapshot an edge was deleted.

3 Analytical Frameworks

Running graph analytics requires dealing with two main aspects – the runtime system components and the interfaces to express the analytical task. Let us discuss the essentials of both.

3.1 Runtime Environment Aspects

In-Memory Graph Layouts. Like the storage and indexing of temporal graphs, dealing with temporal redundancy is important in the in-memory data structures on which analytics are executed. For example, when running a graph algorithm or evaluating a metric on various snapshots of a graph, it might be prohibitively expensive to store all the snapshots in the memory at the same time. However, most graph libraries do require plain isolated version of each graph. The most common solution to this problem is to use an overlaying of multiple versions on one another, and using bitmaps and an additional lookup table to establish the validity of a graph component to one or more versions. This reduces any redundancy of nodes, edges or attributes that occur across multiple versions, yet obtaining access to each snapshot as a static graph through a simple interface. GraphPool [KD13] and Chronos [HML+14] amongst others use variations of this technique. In addition, Chronos also employs such a structure for locality-aware batch scheduling (LABS) of graph computation. More specifically, LABS batches the processing of a vertex across all the snapshots, as well as the information propagation to a neighboring vertex for all the snapshots.

Parallelism. Temporal graph analytics presents opportunities for parallelization in two different ways. First, evaluating a property across multiple snapshots can be easily parallelized, unless . Second, several static graph computation operations such as Pagerank, can be run in parallel themselves. Determining the optimum level of parallelism for an analytical task can be complex and depends on the exact computation steps, nature of graph data and the resources available. TGAF [KD16] provides parallelism by expressing all temporal graphs as RDDs of time evolving nodes or subgraphs and letting the underlying Spark infrastructure plan the parallelism. Leveraging Spark parallelism provides abundant benefits, along with simplicity of design. However, it there are occasions on which certain tasks can be customized for better results. A major drawback with TGAF is the lack of space saving obtained through overlaid structures such as GraphPool. LABS and Llama show effective locality-based multi-core parallelism.

Incremental Computation. Time-range graph analysis can benefits from incremental or shared computation. First, for iterative algorithms such as PageRank, if the target time-range contains a sequence of N snapshots S_0 to S_{N-1}, the result computed on S_0 can be used to initialize the solution for S_1, and so on until S_{N-1}. This reduces the number of iterations needed to converge an algorithm, but also serializes the computations because of the newly created dependence. Chronos utilizes a variation of this principle to effectively perform incremental computation.

3.2 Analytical Interfaces

There are mainly two kinds of historical graph analytical interfaces. The first one, a programmatic interface involves an abstraction of a temporal graph and various operations that can be performed on it, along with calls to several static (sub-)graph routines. The second kind are the visual exploration tools which are usually targeted to more specific types of analytic options but provide an easier means to express the task through GUI interfaces and provide meaningful visualization of the results. One example of a programmatic interface is TGAF [KD16], which abstracts the temporal graph through two main primitives – *Set of temporal nodes (SoN)*, and *Set of temporal subgraphs (SoTS)*. The entire temporal graph or any part of it may be expressed as one of these primitives. A SoN is best illustrated as a three dimensional array

along the axes of time, nodes and attributes. The system also defines graph manipulation operations such as *timeslicing*, *selection*, *filtering*; compute operators such as *node compute map*, *temporal node compute map*, *node compute delta*, amongst others; it also provides analytical operators such as *compare*, *evolution*, and such. A rich set of operators enables the expression of complex analytical operations. We omit further details here and refer the interested reader to [KD16] and [Khu15].

Visualization is an effective means for exploring and analyzing a temporal graph. Following is a rather brief and non-exhaustive reference to techniques and tools that have demonstrated such capabilities. Using visualization of different snapshots, Toyoda et al. [TK05] study events like appearance of pages, relationship between different pages for a historical dataset of Web archive. NetEvViz [KNC+11] extends NodeXL (a popular graph analysis tool), to study evolving networks. It uses an edge color coding scheme for comparative study of network growth. Ahn et al. [ATMS+11] focus on different states of graph components in order to visualize temporal dynamics of a social network. Alternate representations to node-link diagrams have been proposed in order to study temporal nature of social networks, such as TimeMatrix [YEL10] that uses a matrix based visualization to better capture temporal aggregations and overlays. A temporal evolution analysis technique based on capturing visual consistency across snapshots can be seen in the work of Xu et al. [XKHI11]. A survey by Beck et al. [BBDW14] lists several other approaches for visual analysis of dynamic graphs.

References

APS14. Jae-wook Ahn, Catherine Plaisant, and Ben Shneiderman. A task taxonomy for network evolution analysis. *IEEE transactions on visualization and computer graphics*, 20(3):365–376, 2014.

APU09. Sitaram Asur, Srinivasan Parthasarathy, and Duygu Ucar. An event-based framework for characterizing the evolutionary behavior of interaction graphs. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(4):16, 2009.

ATMS+11. Jae-wook Ahn, Meirav Taieb-Maimon, Awalin Sopan, Catherine Plaisant, and Ben Shneiderman. Temporal visualization of social network dynamics: prototypes for nation of neighbors. *Social computing, behavioral-cultural modeling and prediction*, pages 309–316, 2011.

BBDW14. Fabian Beck, Michael Burch, Stephan Diehl, and Daniel Weiskopf. The state of the art in visualizing dynamic graphs. *EuroVis STAR*, 2, 2014.

BCG10. Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2010.

HML+14. Wentao Han, Youshan Miao, Kaiwei Li, Ming Wu, Fan Yang, Lidong Zhou, Vijayan Prabhakaran, Wenguang Chen, and Enhong Chen. Chronos: a graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems*, page 1. ACM, 2014.

KD13. Udayan Khurana and Amol Deshpande. Efficient snapshot retrieval over historical graph data. In *Proceedings of IEEE International Conference on Data Engineering*, pages 997–1008, 2013.

KD16. Udayan Khurana and Amol Deshpande. Storing and analyzing historical graph data at scale. In *Proceedings of International Conference on Extending Database Technology*, pages 65–76, 2016.

Khu15. Udayan Khurana. *Historical Graph Data Management*. PhD thesis, University of Maryland, 2015.

KNC+11. Udayan Khurana, Viet-An Nguyen, Hsueh-Chien Cheng, Jae-wook Ahn, Xi Chen, and Ben Shneiderman. Visual analysis of temporal trends in social networks using edge color coding and metric timelines. In *Proceedings of the IEEE Third Inernational Conference on Social Computing*, pages 549–554, 2011.

KNT10. Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *Link mining: models, algorithms, and applications*, pages 337–357. Springer, 2010.

LKF07. Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

MMMS15. Peter Macko, Virendra J Marathe, Daniel W Margo, and Margo I Seltzer. Llama: Efficient graph analytics using large multiversioned arrays. In *Proceedings of IEEE International Conference on Data Engineering*, pages 363–374, 2015.

NK11. Saket Navlakha and Carl Kingsford. Network archaeology: uncovering ancient networks from present-day interactions. *PLoS Computational Biology*, 7(4), 2011.

PS11. Raj Kumar Pan and Jari Saramäki. Path lengths, correlations, and centrality in temporal networks. *Physical Review E*, 2011.

RLK+11. Chenghui Ren, Eric Lo, Ben Kao, Xinjie Zhu, and Reynold Cheng. On querying historial evolving graph sequences. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2011.

ST99. Betty Salzberg and Vassilis J Tsotras. Comparison of access methods for time-evolving data. *ACM Computing Surveys (CSUR)*, 31(2):158–221, 1999.

TBWK07. Chayant Tantipathananandh, Tanya Berger-Wolf, and David Kempe. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 717–726, 2007.

TK05. Masashi Toyoda and Masaru Kitsuregawa. A system for visualizing and analyzing the evolution of the web with a time series of graphs. In *Proceedings of the sixteenth ACM conference on Hypertext and hypermedia*, pages 151–160, 2005.

TLZN08. Lei Tang, Huan Liu, Jianping Zhang, and Zohreh Nazeri. Community evolution in dynamic multi-mode networks. In *Proceedings of the 14th ACM SIGKDD International conference on Knowledge discovery and data mining*, pages 677–685. ACM, 2008.

XKHI11. Kevin S Xu, Mark Kliger, and Alfred O Hero III. Visualizing the temporal evolution of dynamic networks. *stress (X)*, 1:2, 2011.

YEL10. Ji Soo Yi, Niklas Elmqvist, and Seungyoon Lee. Timematrix: Analyzing temporal social networks using interactive matrix-based visualizations. *Intl. Journal of Human–Computer Interaction*, 26(11-12):1031–1051, 2010.