Towards sequenced semantics for evolving graphs

Vera Zaychik Moffitt
Drexel University
zaychik@drexel.edu

Julia Stoyanovich*
Drexel University
stoyanovich@drexel.edu

ABSTRACT

The research community has adopted a sequence of snapshots as the logical representation of evolving graphs — graphs that change over time and whose history of evolution we want to preserve for analysis. This paper argues that the snapshot sequence model of evolving graphs is insufficient for representation and analysis of a wide range of networks. Instead, we propose to use the interval model with sequenced semantics. In this model nodes and edges are associated with their validity intervals, and operations adhere to the properties of snapshot reducibility, extended snapshot reducibility, and change preservation. We show the advantages of adopting this model for evolving graphs and lay the groundwork for an evolving graph query language with sequenced semantics. We also discuss several challenges of efficiently supporting sequenced semantics in a distributed setting.

1. INTRODUCTION

Evolving graphs are used to represent a wide range of phenomena, including the Web, social networks, communication and transportation networks, interaction networks, metabolism pathways, and many others. Researchers study graph evolution rate and mechanisms, impact of specific events on further evolution, spatial and spatio-temporal patterns, and how graph properties change over time.

The dominant logical model for evolving graphs over the past 20 years has been a sequence of static graphs, termed *snapshots*. This model is a graph-specific adaptation of the *point-based temporal model* [19], and it introduces a semantic ambiguity that has been well studied in the temporal relational databases literature [2]: if an entity (graph, vertex or edge) with the same attributes exists in two consecutive snapshots, does it represent the same fact or two different facts? What does it mean for an entity to change?

*This work was supported in part by NSF Grants No. 1464327 and 1539856, and BSF Grant No. 2014391.

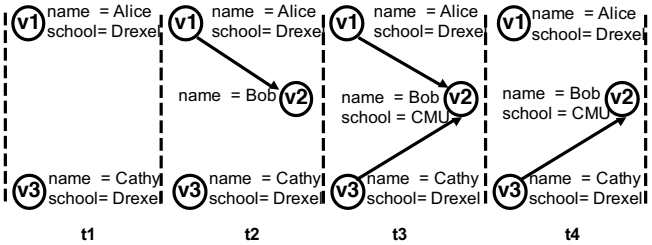


Figure 1: A social network as a snapshot sequence.

Figure 1 shows an example of an evolving social network, in which vertices represent people, while edges represent interactions between them such as likes and conversations. In this example, did Alice and Bob have two conversations over the time period $[t1, t4)$ or one long one? Did Alice undergo any changes during this time? Which user was the most active in this network, as defined by the number of distinct interactions? What is the rate of change of this network? We cannot answer these questions without additional information in a point-based model. Suppose that Alice held a temporary position at Drexel at time $t1$ and transferred to a permanent one at time $t2$. This information cannot be represented in the point-based model. Suppose that Alice and Bob had two short interactions, while Cathy and Bob had one longer one. The point-based model cannot distinguish between these two cases.

This kind of semantic ambiguity affects several graph operations, most notably aggregation and retrieval of change history, and, as a result, local (confined to specific entity or subset of entities) and global (whole-graph) temporal queries that are useful for evolving graph analysis.

In a point-based model [19] each entity is time-stamped with its validity time. For practical reasons, intervals are often used as syntactic abbreviations for sets of points. To use intervals in a time-stamped model, we coalesce, i.e., merge value-equivalent tuples over overlapping and adjacent time points [1]. Importantly, the use of intervals to represent a sequence of value-equivalent time-adjacent snapshots is not semantically equivalent to a model with *sequenced semantics*, where entities are time-stamped with intervals that have meaning. A work-around to avoid coalescing tuples that represent different facts is to add attributes to entities, in order to distinguish between changes and non-changes. For example, we can add position title to the vertex Alice to state that Alice changed jobs at time $t2$, and add a conversation id to each edge to designate distinct conversations. Unfortunately, this solution is ad-hoc rather than general and does not hold up over time, as discussed in [2].

We contend that a snapshot sequence model of evolving graphs is insufficient for representation of a wide range of networks and propose instead to use the interval model with sequenced semantics. Facts in our model correspond to graph vertices and edges, rather than to graph snapshots in their entirety (as in Figure 1). This representational choice is orthogonal to the issue of point-based vs. sequenced semantics, but has an important advantage. Many evolving graph queries include temporal predicates over vertices or edges, e.g., compute a subgraph containing only vertices that persist for at least a year. Such queries cannot be evaluated directly over a sequence of snapshots.

In Section 2 we briefly survey existing models and summarize relevant work in temporal databases. We then propose a new model in Section 3. In Section 4 we discuss the challenges of efficient computation under sequenced semantics in a distributed environment. We conclude with future research directions in Section 5.

2. RELATED WORK

Evolving graph models. While temporal models in the relational literature are very mature, the same cannot be said about the evolving graphs literature. Evolving graph models differ in what time stamp they use (point or interval stamping), what top-level entities they model (graphs or sets of nodes and edges), whether they represent topology only or attributes or weights as well, and what types of evolution are allowed. All evolving graph models require node identity, and thus edge identity as well, to persist across time. See [20] for a survey of evolving graph models.

The first mention of evolving graphs that we are aware of is by Harary and Gupta [6] who informally proposed to model the evolution as a sequence of static graphs. This model has been predominant in the research literature ([4, 7, 14] and many others), with various restrictions on the kinds of changes that can take place during graph evolution. For example, Khurana and Deshpande [7] use this model with the restriction that a node, once removed, cannot reappear. In [4] and [14] there is no notion of time, only a sequence of graphs. It is important to note that we are talking about the logical model of the evolving graphs, rather than a physical representation. For example, Semertzidis et al. [16] present a concrete representation of an evolving graph called VersionGraph that is similar to the logical model we propose here, yet their logical model is still a sequence of snapshots.

The advantages of the snapshot sequence model are that (a) it is simple and (b) if snapshots are obtained by periodic sampling, which is a very common approach, it accurately represents the states of the graph at the sampled points without making assertions about unknown times. For example, the WWW is so large that it is impossible to create a fully accurate snapshot that represents any moment in time. An important limitation of this model, in addition to the semantic ambiguity on what constitutes a change, is that it forces a specific time granularity, whereas open-closed time intervals can be broken down into any desired level of granularity.

Temporal relational models. The question of semantics of temporal data has been thoroughly explored in the relational temporal database community. Böhlen et al. [2] defined point and sequenced models, and showed that the difference between the models lies in the properties of the operators, and not in the use of intervals as representational devices. With this foundation, Dignös et al. [3] defined sequenced semantics, with properties of snapshot reducibility, extended snapshot reducibility, and change preservation. Snapshot reducibility means that a temporal operator produces the same result as an equivalent non-temporal operator over corresponding snapshots. Extended snapshot reducibility allows references to timestamps in the operators by propagating them as data. Point semantics has both of these properties as well.

The third property, change preservation, is unique to sequenced semantics. It states that operators only merge contiguous time points of a result if they have the same lineage. As shown in [3], all three properties can be guaranteed through the use of the normalize and align operators on non-temporal relations, extended with an explicit time attribute.

As we as a community move to more and more sophisticated analyses of evolving graphs, we need to adopt the state of the art in temporal databases.

3. DATA MODEL

We now describe the *logical* representation of an evolving graph, called a TGraph. A TGraph represents a single graph, and models evolution of its topology and of vertex and edge attributes.

Following the SQL:2011 standard [8], a period (or interval) $\boldsymbol{p} = [s, e)$ represents a discrete set of time instances, starting from and including the start time s, continuing to but excluding the end time e. Time instances contained within the period have limited precision, and the time domain has total order. In the rest of this paper we use the terms interval and timestamp interchangeably.

A TGraph is represented with four temporal SQL relations [1], and uses sequenced semantics [3], associating a fact (existence of a vertex or edge, and an assignment of a value to a vertex or edge attribute) with an interval.

A snapshot of a temporal relation R, denoted $\tau_c(R)$ is the state of R at time point c.

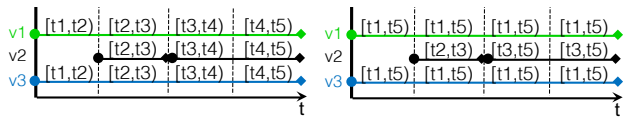
We use the property graph model [15] to represent vertex and edge attributes: each vertex and edge during period $\boldsymbol{p}$ is associated with a (possibly empty) *set* of properties, and each property is represented by a key-value pair. Property values are not restricted to be of atomic types, and may, e.g., be sets, maps or tuples.

We now give a formal definition of a TGraph, which builds on the model of [12] and is adjusted to support sequenced semantics.

DEFINITION 3.1 (TGRAPH). *A TGraph is a pair* $\mathsf{T} = (\mathsf{V}, \mathsf{E})$. *V is a valid-time temporal SQL relation with schema* $\mathsf{V}(\underline{v}, \underline{\boldsymbol{p}})$ *that associates a vertex with the time period during which it is present. E is a valid-time temporal SQL relation with schema* $\mathsf{E}(\underline{v_1}, \underline{v_2}, \underline{\boldsymbol{p}})$, *connecting pairs of vertices from V.* T *optionally includes vertex and edge attribute relations* $\mathsf{A}^{\mathsf{V}}(\underline{v}, \underline{\boldsymbol{p}}, a)$ *and* $\mathsf{A}^{\mathsf{E}}(\underline{v_1}, \underline{v_2}, \underline{\boldsymbol{p}}, a)$, *where a is a nested attribute consisting of key-value property pairs. Relations of* T *must meet the following requirements:*

R1: Unique vertices/edges *In every snapshot* $\tau_c(\mathsf{V})$ *and* $\tau_c(\mathsf{E})$ *a vertex/edge exists at most once.*

R2: Unique attribute values *In every snapshot* $\tau_c(\mathsf{A}^{\mathsf{V}})$ *and* $\tau_c(\mathsf{A}^{\mathsf{E}})$, *a vertex/edge is associated with at most one attribute (which is itself a set of key-value pairs representing properties).*



(a) with fragments (b) with full timestamps

Figure 2: Vertices from Figure 1 split in 4 partitions.

R3: Referential integrity *In every snapshot* $\tau_c(\mathsf{T})$*, foreign key constraints hold from* $\tau_c(E)$ *(on both* v_1 *and* v_2*) and* $\tau_c(\mathsf{A}^{\mathsf{V}})$ *to* $\tau_c(V)$*, and from* $\tau_c(\mathsf{A}^{\mathsf{E}})$ *to* $\tau_c(E)$*.*

Requirements **R1, R2, R3** guarantee soundness of the TGraph data structure, ensuring that every snapshot of a TGraph is a valid graph. Graphs may be directed or undirected. For undirected graphs we choose a canonical representation of an edge, with $v_1 \leq v_2$ (self-loops are allowed). At most two edges can exist between any two vertices at any time point, one in each direction.

Definition 3.1 presents a *logical* data structure that admits different physical representations, including, e.g., a columnar representation (each property in a separate relation, supporting different change rates), by a hash-based representation of [18], or in some other way. The logical model also allows for distributed storage in HDFS.

4. SEQUENCED SEMANTICS IN A DISTRIBUTED ENVIRONMENT

Many interesting static graphs are so large that they necessitate a distributed approach, as evidenced by the plethora of works on Pregel-style computation and graph partitioning [10]. In this section we discuss the challenges inherent in supporting three properties of sequenced semantics — snapshot reducibility, extended snapshot reducibility, and change preservation — in a distributed environment.

Snapshot reducibility. Evolving graphs can be partitioned among the available machines using time locality. Following convention, we refer to the operator that can produce such partitioning as a *splitter*. The splitter places each tuple (vertex or edge) into one or more partitions based on its timestamp. The goal of the splitter is to form partitions that are balanced, i.e., have approximately the same number of items, under the assumption that most operations can be executed locally at each partition. Recall that snapshot reducibility requires a temporal operator to produce the same result as if it were evaluated over each snapshot. Validity period of a tuple that spans more than one temporal partition is split, and the tuple is replicated across partitions. This increases the overall size of the relation, but all operations can now be carried out within each partition. See Figure 2a for a simple example of the V relation being split into four temporal partitions.

For the purposes of illustration, consider the temporal subgraph operation, a generalization of subgraph matching for non-temporal graphs [12]. Temporal vertex-subgraph $\mathsf{sub}_v^T(q_v^t, \mathsf{T}) = \mathsf{T}'(\mathsf{V}', \mathsf{E}', \mathsf{A}^{\mathsf{V}'}, \mathsf{A}^{\mathsf{E}'})$ computes an induced subgraph of T, with vertices defined by the temporal conjunctive query q_v^t. Note that this is a subgraph query, and so $\mathsf{V}' \subseteq^T \mathsf{V}$. Observe that we can carry out the subgraph operation with non-temporal predicates, e.g., name='Alice', at each partition individually, without any cross-partition communication.

The question of optimal splitting has been addressed by Le et al. [9], who demonstrated that a temporal relation can be efficiently split into k buckets in cases of both internal memory and external memory, and guarantee optimality of the solution. This method requires a sequential scan of the relation to compute an index called the stabbing count array. How to make this method more efficient in a distributed environment is an open question.

The subgraph operation requires co-partitioning of graph relations to enforce referential integrity on edges. A number of alternatives for co-partitioning present themselves, as the vertex, edge and attribute relations are not guaranteed to have the same splitters due to different evolution rates. Typically, vertices are co-partitioned with edges in the non-temporal case [5], and this likely is most efficient with evolving graphs as well.

Extended snapshot reducibility. Snapshot reducibility can be guaranteed in the distributed setting, as shown above for a subgraph query without temporal predicates. In general, a subgraph query q_v^t may use any of the constituent relations of T, and may explicitly reference temporal information in compliance with the extended snapshot reducibility property of sequenced semantics. Refer back to Figure 2a and assume time granularity of years. If we perform the subgraph operation, selecting vertices that persist for longer than 2 years, over the split then we will get no matches. However, the original relation contains two matches – only Bob does not meet the predicate. To support extended snapshot reducibility over a split relation, during partitioning tuples should be placed into their partitions with their full original timestamps. Incidentally, this is what Le at al. describe in their work on optimal splitters [9]. Figure 2b shows relation V split in the same four partitions with this approach.

Change preservation. Change preservation property requires that derived tuples should only be coalesced if they share lineage. To support this property, normalize and align operators are used [3]. The normalize operator splits each tuple in the input relation w.r.t. a group of tuples such that each timestamp fragment is either fully contained or disjoint with every timestamp in the group. The align operator splits each tuple w.r.t. a group of tuples such that each timestamp fragment is either an intersection with one of the tuples in the group or is not covered by any tuple in a group.

The normalize operator splits each tuple w.r.t. to a group defined by the operation. For example, consider the attribute-based node creation operation on graphs [12], an operation similar to aggregation on temporal relations. This operation allows the user to generate a TGraph in which vertices correspond to disjoint groups of vertices in the input that agree on the values of all grouping attributes. For instance, $\mathsf{node}_a^T(\mathsf{school}, \mathsf{T})$ will compute a vertex for each value of $\mathsf{A}^{\mathsf{V}}.\mathsf{a.school}$. While the group defined for each tuple (distinct value of school) spans temporal partitions, only tuples within the same partition overlap. Thus, the normalize and align operations can be carried out locally at each partition.

An important challenge to address is how to efficiently support aggregation over temporal windows in a distributed setting. This operation requires cross-partition communication, which impacts the cost model, requiring a generalization of the approach of Le et al. [9].

Partitioning of evolving graphs. Large evolving graphs present additional challenges compared to static graphs and

Table 1: Connected components and PageRank with different temporal partitioning, seconds.

(a) wiki-talk

width	k	CC	PR
8	23	382	1,086
16	12	328	1,037
bal.	16	351	720
bal.	24	408	375

(b) nGrams

width	k	CC	PR
8	26	1,324	2,867
16	13	856	1,873
bal.	3	321	740
bal.	16	422	519

temporal relations alone. Each graph snapshot may be too large to fit into a single partition. This necessitates partitioning graphs by both time and structure. Miao et al. [11] have demonstrated within their ImmortalGraph system that different locality, structural or temporal, is more appropriate for different graph queries. However, their results do not directly translate to the distributed environment. Miao et al. showed that spatial locality provides better performance than temporal locality in global point queries, i.e., queries that compute over a snapshot corresponding to a particular time point. In a distributed setting we do not expect these results to hold, since communication costs generally dominate the overall performance, and partitioning by time alone will guarantee that a snapshot is distributed among the lowest number of partitions.

Global range queries such as change in graph centrality over time, on the other hand, are computed over multiple snapshots and their performance depends on the method of computation. We can utilize temporal locality and compute on each snapshot independently. Assuming that each partition fits one or more snapshots, the maximum number of snapshots across all partitions will determine the overall performance. With structural locality we can distribute edges across the partitions using any of the already proposed partitioning approaches such as range- and hash-based [17] or EdgePartition2D (E2D).

We explored the effectiveness of partitioning strategies in graphs that undergo changes in topology over time, and found that structural partitioning such as in ImmortalGraph is effective only when graph topology changes very little [13]. We found that a hybrid approach that combines temporal and structural locality is promising. We are currently investigating methods for selecting a partitioning strategy that provides the best overall performance.

Preliminary experiments. We conducted some preliminary experiments to see the effect of temporal partitioning on distributed execution of analytics, which present one of the heaviest computational workloads. PageRank and Connected components analytics were executed on the wiki-talk[1] and nGrams[2] datasets.

Wiki-talk contains 179 time periods. nGrams contains over 400, but we used the first 208. Both datasets exhibit strong skew, with few edges at the start of the datasets and increasing by several orders of magnitude towards the end. We compared equi-width and equi-depth temporal partitioning, using 8 and 16 consecutive intervals for equi-width, and using offline optimal split of edges with varying number of splitters k. Each dataset was partitioned first temporally, and then spatially using Edge2D partitioning. Table 1 shows that equi-depth partitioning is superior to equi-width in all cases but one. However, the number of splitters is key in obtaining good results. We are currently investigating this phenomenon further.

[1]http://dx.doi.org/10.5281/zenodo.49561

[2]http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

5. CONCLUSION

We have argued that modeling evolving graphs using snapshot sequences presents semantic difficulties. As an alternative, we proposed a vertex-edge model with sequence semantics and discussed several challenges of supporting this model in a distributed setting. We are currently working on implementing this model in Apache Spark, and exploring the performance of different physical representations and partition strategies.

6. REFERENCES

[1] M. H. Böhlen et al. Coalescing in temporal databases. In *VLDB*, 1996.

[2] M. H. Böhlen et al. Point versus interval-based temporal data models. In *ICDE*, 1998.

[3] A. Dignös et al. Temporal alignment. In *SIGMOD*, 2012.

[4] A. Fard et al. Towards efficient query processing on massive time-evolving graphs. In *IEEE CollaborateCom*, 2012.

[5] J. E. Gonzalez et al. GraphX: Graph processing in a distributed dataflow framework. In *OSDI*, 2014.

[6] F. Harary and G. Gupta. Dynamic graph models. *Mathematical and Computer Modelling*, 25(7), 1997.

[7] U. Khurana and A. Deshpande. Efficient snapshot retrieval over historical graph data. In *ICDE*, 2013.

[8] K. G. Kulkarni and J. Michels. Temporal features in SQL: 2011. *SIGMOD Record*, 41(3), 2012.

[9] W. Le et al. Optimal splitters for temporal and multi-version databases. In *SIGMOD*, 2013.

[10] R. R. McCune et al. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM CSUR*, 1(1), 2015.

[11] Y. Miao et al. ImmortalGraph: A system for storage and analysis of temporal graphs. *ACM TOS*, 11(3), 2015.

[12] V. Z. Moffitt and J. Stoyanovich. Querying evolving graphs with portal. Dec. 2016. arXiv:1602.00773.

[13] V. Z. Moffitt and J. Stoyanovich. Towards a distributed infrastructure for evolving graph analytics. In *TempWeb*, 2016.

[14] C. Ren et al. On querying historical evolving graph sequences. *PVLDB*, 4(11), 2011.

[15] I. Robinson, J. Webber, and E. Eifrem. *Graph databases*. O'Reilly Media, Inc., 2013.

[16] K. Semertzidis et al. TimeReach: Historical reachability queries on evolving graphs. In *EDBT*, 2015.

[17] J. Seo et al. Distributed socialite: A datalog-based language for largescale graph analysis. *PVLDB*, 6(14), 2013.

[18] W. Sun et al. SQLGraph: An efficient relational-based property graph store. In *SIGMOD*, 2015.

[19] D. Toman. Point-stamped temporal models. In *Encyclopedia of Database Systems*. 2009.

[20] A. Zaki et al. Comprehensive survey on dynamic graph models. *IJACSA*, 7(2), 2016.