Temporal Graph Algebra

Vera Zaychik Moffitt
Drexel University, USA
zaychik@drexel.edu

Julia Stoyanovich*
Drexel University, USA
stoyanovich@drexel.edu

ABSTRACT

Graph representations underlie many modern computer applications, capturing the structure of such diverse networks as the Internet, personal associations, roads, sensors, and metabolic pathways. While analysis of static graphs is a well-explored field, new emphasis is being placed on understanding and representing the ways in which networks change over time. Current research is delving into graph evolution rate and mechanisms, the impact of specific events on network evolution, and spatial and spatio-temporal patterns. However, systematic support for evolving graph querying and analytics still lacks. Our goal is to fill this gap, giving users an ability to concisely express a wide range of common analysis tasks.

In this paper we combine advances in graph databases and in temporal relational databases and propose an evolving graph model, including a representation called TGraph and an algebra called TGA, that adheres to point-based semantics. TGA includes principled temporal generalizations of conventional graph operators, as well as novel operators that support exploratory analysis of evolving graphs at different levels of temporal and structural granularity.

CCS CONCEPTS

•Information systems → Temporal data; Graph-based database models; Query languages;

KEYWORDS

Evolving Graphs; Analytical Evolutionary Analysis; Point-based Models

ACM Reference format:
Vera Zaychik Moffitt and Julia Stoyanovich. 2017. Temporal Graph Algebra. In *Proceedings of DBPL 2017, Munich, Germany, September 1, 2017,* 12 pages.
DOI: 10.1145/3122831.3122838

1 INTRODUCTION

Analysis of evolving graphs is a rich new area of research that has been getting increasing attention [2, 29, 32, 33]. This is a natural consequence of the prevalence of graph datasets that represent phenomena that change over time, such as social networks and the Web, and where evolution provides an important dimension of interest from which to draw insight.

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

DBPL 2017, Munich, Germany

DOI: 10.1145/3122831.3122838

Given the progress in both graph databases and temporal databases, it is expected that evolving graphs would be supported as well, being at the intersection of the two areas. And yet, systematic support for evolving graph querying and analytics still lacks, despite much recent interest and activity, and despite increased variety and availability of evolving graph data. The goal of our work is to fill this gap.

The current de facto standard representation of evolving graphs is the snapshot sequence, where a state of a graph is associated with a time point, or an interval, during which the graph was in that state [7, 9, 13, 14, 20, 21, 26, 27, 32–34, 37]. This representation is conceptually simple, and it naturally supports a particular class of operations — those that are applied to each state of the graph. For example, we can compute answers to subgraph or reachability queries within each snapshot. This kind of processing adheres to the principle of *snapshot reducibility*, which states that the result of applying a temporal operator to a database is equivalent to applying the non-temporal variant of the operator to each state of the database [6].

The fundamental disadvantage of using the snapshot sequence as the conceptual representation is that it restricts the range of operations that can be expressed. Suppose that we wish to retrieve nodes connected by a journey: a temporally consecutive non-decreasing path [8, 14]. This query cannot be expressed over a snapshot sequence, and neither can any other query that makes explicit reference to temporal information associated with nodes, edges or their properties. Semantics of operations that make explicit references to time are formalized in the temporal relational literature as the principle of *extended snapshot reducibility*, where timestamps are made available to operators by propagating time as data [6]. This principle allows users to reference time explicitly in predicates, but not manipulate time directly.

In summary, a model in which the snapshot sequence is the conceptual representation can support snapshot reducibility, but it lacks extended snapshot reducibility for operators in which temporal predicates are stated over nodes, edges and node / edge properties. Snapshot reducibility and extended snapshot reducibility are two required properties of temporal models with *point-based semantics* [6], which we aim to support.

In this paper, we propose a representation of evolving graphs that is based on extending property graphs [3] with time, along with a set of algebraic operations. Our model is based on three central principles of temporal data management: (1) that time should be stored explicitly in the model and treated differently from regular data; (2) that temporal operations are intuitive generalizations of non-temporal operations, with implicit handling of time; and (3) that additional expressive power can be gained by allowing explicit access to time data by operations that are temporal in nature.

Contributions and roadmap. The rest of this paper is organized as follows. We propose a conceptual representation of an

evolving graph called TGraph, which can be viewed as a temporal generalization of the property graph model [3] and captures the evolution of both graph topology, and of node and edge attributes. TGraph is described in Section 2.

To provide systematic support for evolving graph analysis, we propose a compositional temporal graph algebra called TGA. Our model adheres to point-based semantics [6], and TGA operations provide a principled temporal generalization of conventional graph operations under this semantics. Further, we propose novel operations that support exploratory analysis of evolving graphs at different levels of temporal and structural granularity. TGA is described in Section 3. We present formal properties of TGA, with a focus on its temporal completeness, in Section 4.

Our goal is to define an algebra that has clear semantics and is sufficiently expressive to support evolving graph analysis for a wide audience of data scientists and researchers. We illustrate use cases of TGA in Section 5.

We present related work in Section 6 and conclude in Section 7.

2 TEMPORAL GRAPH MODEL

Our focus is on analytical evolutionary analysis of graphs, where the goal is to model, quantify and understand the changes that have occurred in the underlying graph over time. This is complementary to maintenance methods (or streaming), where the goal is to maintain the result of some query or mining process continuously over time [2]. Because of our interest in analysis over time, we assume that the graph is fully evolved and consider valid time, as opposed to transaction time or bi-temporal representations [24].

2.1 Preliminaries

We assume a linearly ordered discrete time domain Ω^T where time *instances* (or time points) have limited precision. In temporal relational databases, a *valid-time* temporal relation schema is represented as $R(a_1, \ldots, a_m \mid T)$, where $a_1, \ldots, a_m$ are non-temporal attributes and T is a temporal attribute over $\Omega^T \times \Omega^T$. The timestamp attribute T is special and thus separated by a | symbol in the signature of R. Following the SQL:2011 standard [24], we use closed-open intervals: an interval represents a discrete contiguous set of time instances from domain Ω^T, starting from and including the start time, continuing to but excluding the end time.

Associating a tuple with a time interval during which the fact represented by the tuple is known to hold is called *tuple timestamping* [31]. In principle, T can be a single time point or a set of time points. We use periods of validity to compactly represent the constituent time points. This is a common representation technique, which does not add expressive power to the data model, compared to associating each tuple with a single time instant [10].

With point-based semantics, a relation is required to be temporally coalesced: A pair of value-equivalent temporally adjacent tuples should be stored as a single tuple, and this property should be maintained through operations [4]. Requiring relations to be coalesced is both space-efficient and avoids semantic ambiguity [19].

A *snapshot* of a temporal relation $R(a_1, \ldots, a_m \mid T)$ at time point $p \in \Omega^T$, denoted $\tau_p(R)$, is a non-temporal relation with schema $(a_1, \ldots, a_m)$ that represents the state of R at time p.

2.2 TGraph

We now describe the logical representation of an evolving graph, called a TGraph, which represents a single graph, and models the evolution of its topology and of vertex and edge properties. TGraph is a multi-graph: its nodes and edges have identity, and multiple edges may connect a given pair of nodes.

We now give a formal definition of a TGraph. This definition extends the static property graph definition of Angles et al. [3] by associating periods of validity with graph nodes, edges and property values.

Definition 2.1 (TGraph). A TGraph is a 6-tuple $\mathcal{G} = (V, E, L, \rho, \xi^T, \lambda^T)$, where:

- V is a finite set of *nodes*, E is a finite set of *edges*, $V \cap E = \emptyset$, and L is a finite set of property labels;
- $\rho : E \rightarrow (V \times V)$ is a total function that maps an edge to its source and destination nodes;
- $\xi^T : (V \cup E) \times \Omega^T \times \Omega^T \rightarrow B$ is a total function that maps a node or an edge and time period to a Boolean, indicating existence of the node or edge during the time period; and
- $\lambda^T : (V \cup E) \times L \times \Omega^T \times \Omega^T \rightarrow val$ is a partial function that maps a node or an edge, a property label, and a time period to a value of the property during the time period.

The following additional conditions hold on λ^T: A property can only take on a value during the time period when the corresponding node or edge exists. The property set of a node or an edge may **not** be empty at any time point when the node or edge exists. The domain of values for a property is either atomic or a V-relation [1] (a restricted kind of a nested relation). For simplicity, we assume that all atomic properties draw values from a single domain.

Note that the function ρ in Definition 2.1 is not temporal — an edge always connects the same two nodes, whenever it exists. As in temporal relational models, time is part of the TGraph model, and is not simply encoded as one of the properties. We will see in Section 3 that time is treated differently from data in operations — it can be accessed in predicates but cannot be modified directly.

We now provide an alternative TGraph definition using a pair of nested temporal relations. In Appendix A we show that the models in Definitions 2.1 and 2.2 are equivalent, by providing a bijection between them. We present an alternative TGraph definition because it is sometimes more convenient to define the semantics of an operation over a pair of relations rather than over a property graph.

Definition 2.2 (TGraph-Relational). A TGraph is a pair $\mathcal{G} = (\mathsf{TV}, \mathsf{TE})$, where:

- $\mathsf{TV}(\underline{v}, a : (l, val) \mid T)$ is a valid-time nested temporal relation that associates a vertex and its attribute a with the time period during which the vertex is present and its properties are unchanged. Attribute a is a set of property-value pairs, where l is a property label and val is a property value.
- $\mathsf{TE}(\underline{e}, v_1, v_2, a : (l, val) \mid T)$ is a valid-time nested temporal relation, in which each edge connects a pair of nodes from TV, and is associated with an attribute a, a set of property-value pairs.

In both TV and TE, a is non-empty, and contains a non-temporal nested relation, specifically, a V-relation [1]. For simplicity, we assume that all atomic properties draw values from a single domain.

Table 1: A co-authorship network represented using the TGraph model, consisting of two nested temporal relations.

TV			
v	**a**	**T**	
v1	type=p, name=Alice, school=Drexel	['15/01,'15/07)	
v2	type=p, name=Bob	['15/02,'15/05)	
v2	type=p, name=Bob, school=CMU	['15/05,'15/10)	
v3	type=p, name=Cathy, school=Drexel	['15/01,'15/10)	

TE				
e	**v1**	**v2**	**a**	**T**
e1	v1	v2	type=co-author, cnt=3	['15/02,'15/06)
e2	v2	v3	type=co-author, cnt=4	['15/07,'15/10)

Relations of $\mathcal{G}$ must meet the following requirements:

R1: Unique vertices/ edges. In every snapshot τ_p(TV) and τ_p(TE), where p is a time point, a vertex or an edge exists at most once. This corresponds to set-based semantics with duplicate-free temporal relations.

R2: Referential integrity. In every snapshot τ_p(TE) foreign key constraints hold to τ_p(TV) on both v_1 and v_2. That is, every edge at time p connects a pair of vertices that also exist at time p.

R3: Coalesced. Value-equivalent tuples in TV and TE with consecutive or overlapping time periods are merged. For the nested attribute a, value equivalence is interpreted as standard set equivalence.

R4: Required property. For any $v \in \mathsf{TV}, v.a \neq \emptyset$, and for any $e \in \mathsf{TE}, e.a \neq \emptyset$. That is, we require that each node and edge have at least one property in its property set.

R5: Constant edge association. For any pair of edges $e_1, e_2 \in \mathsf{TE}, e_1.e = e_2.e \implies e_1.v_1 = e_2.v_1 \wedge e_1.v_2 = e_2.v_2$. That is, consistent with the non-relational definition, we require that an edge always connects the same two nodes, whenever it exists.

Note that condition **R5** is not redundant although e is the key of TE. This is because, in temporal models, the key alone does not determine the values of non-key attributes, it only does so in combination with the timestamp T.

Table 1 gives an example of a TGraph that shows evolution of a co-authorship network. Vertex v1 persists without change from '15/01 to '15/07, but vertex v2 has a property change event at '15/05, thus creating a new tuple in TV. For the sake of readability we display the nested attribute a as a list of key value pairs.

Requirements **R1** and **R2** guarantee soundness of the TGraph model, ensuring that every snapshot of TGraph represents a valid (non-temporal) graph. If we remove requirement **R1**, a snapshot at some point p may contain two instances of the same node or edge, which breaks set-based graph semantics. Requirement **R2** prevents a situation where an edge connects a node that does not exist at that time instant. Requirement **R3** avoids semantic ambiguity and ensures correctness of algebraic operations in point-based temporal models such as ours [19]. Requirement **R4** provides compatibility with the graph-based definition, where at least one property is required, and prevents loss of information when relations TV and TE are un-nested during operations.

In Appendix A we show that the models in Definitions 2.1 and 2.2 are equivalent, by providing a bijection between them. We now proceed to define operations of our temporal graph algebra, using either TGraph or TGraph-Relational, as convenient.

3 TGA — TEMPORAL GRAPH ALGEBRA

This section defines our proposed temporal graph algebra (TGA). Our goal is to enable users to express a wide range of analysis tasks, with a focus on analysis over time.

TGA adheres to point-based semantics: time intervals associated with a node or an edge indicate the validity of the associated values, without regard for the original user-entered validity periods [19]. That is, two value-equivalent tuples with overlapping or consecutive intervals are coalesced, per Definition 2.2 (**R3**). As is desired with point-based models, TGA operations have the snapshot reducibility and extended snapshot reducibility properties [12]. We described these properties in the introduction, and will analyze them formally in Section 4.

The semantics of each operator is specified either by a translation into a sequence of (nested) temporal relational algebra (TRA) operators [12], or by applying navigational patterns [3], extended with time. We assume that model properties are maintained by the temporal relational model, e.g., that if a TRA operator is known to produce uncoalesced results, they are coalesced in the final output, and that the integrity constraints from TE to TV are enforced. We use the foreign key constraint enforcement method that allows the operation and then modifies the output to remove tuples from TE for which a corresponding vertex tuple does not exist in TV, or to trim the period of validity of an edge to correspond to the periods of validity of its vertices. For readability, we use a shorthand notation and refer to $\mathcal{G}$.TV as TV and to $\mathcal{G}$.TE as TE, when $\mathcal{G}$ is clear from context.

3.1 Maintaining identity equivalence

Tuples in the conventional (non-graph) temporal relational model have no identity and are coalesced only if they are value-equivalent. This is in contrast to the TGraph model, which represents an evolving graph by a pair of nested temporal relations, but requires both vertex and edge identity to persist through time and propagate through operations. We refer to vertex and edge tuples that have the same identifiers as identity-equivalent:

Definition 3.1 (Identity-equivalent). Two vertex tuples in TV, resp. edge tuples in TE, are identity-equivalent if they agree on the value of the key attribute, $\underline{v}$ for TV and $\underline{e}$ for TE, regardless of the values of the non-key attributes.

To produce a valid TGraph, we need to output valid *keyed* relations TV and TE with no identity-equivalent vertices or edges over overlapping time instants. TRA does not have a mechanism for coalescing based on key only. In addition, TV and TE are *nested* relations, since each node and edge attribute is a set of key-value pairs. Any TGA operation that may produce multiple identity-equivalent tuples over overlapping time periods requires an additional set of aggregation functions, to be used in the *resolve primitive*, which we now define.

Table 2: $\mathbf{trim}^T_{['15/05,'15/08)}(\mathcal{G})$

TV			
v	**a**	**T**	
v1	type=p,name=Alice,school=Drexel	['15/05,'15/07)	
v2	type=p,name=Bob,school=CMU	['15/05,'15/08)	
v3	type=p,name=Cathy,school=Drexel	['15/05,'15/08)	
TE			
e	**v1**	**v2**	**a**	**T**
e1	v1	v2	type=co-author,cnt=3	['15/05,'15/06)
e2	v2	v3	type=co-author,cnt=4	['15/07,'15/08)

Definition 3.2 (Resolve primitive). Consider a nested temporal relation $R(\underline{k}, a \mid T)$, with property labels $l_i, \ldots, l_n$, and aggregation functions $f_i, \ldots, f_n$. The resolve primitive computes:

$$\mathsf{res}(f_1(l_1), \ldots, f_n(l_n), R) = \nu^T_{a,l,val}\Big(\bigcup^i {}_{k,l}\gamma^T_{f_i(val)}(\sigma^T_{l=l_i}(\mu^T_a(R)))\Big).$$

In Definition 3.2, $\mu^T_a(R)$ is the temporal unnesting operator that produces an intermediate relation with schema $(k, l, val \mid T)$. We must now create a group for each pair of key k and label l_i, and apply the label-specific aggregation function f_i to the values val in the group. Because different labels require different aggregation functions, the result cannot be computed by a single group by. Instead, we process each group separately, as follows. For each label l_i, compute temporal selection with the condition $l = l_i$, followed by temporal aggregation, applying function $f_i(val)$ to the values that correspond to key k and label l_i. The results of temporal aggregation over all i labels are combined with the union operator. Finally, the nesting operator ν^T is applied to nest l, val back into a.

In summary, resolve unnests the input relation, computes a temporal aggregation by key, applying the specified aggregate function to each property, and then nests to produce the final result consistent with the TGraph model. The resolve primitive is used by several TGA operators to maintain model integrity.

3.2 Trim

It is often useful to analyze a portion of the overall TGraph history. The trim operator computes a new TGraph, limited only to those nodes and edges that existed during the specified period.

Example 3.3. Consider the example TGraph $\mathcal{G}$ in Table 1, which we use throughout this section. trim with an input period ['15/05,'15/08) computes $\mathcal{G}'$ as depicted in Table 2. Observe that vertex $v2$ is not present in $\mathcal{G}'$ because it is wholly outside of the input period. Observe also that the period of validity of $v1$ has been trimmed — modified to equal the intersection with the input period.

Definition 3.4 (Trim). The trim operator, denoted $\mathsf{trim}^T_{\boldsymbol{c}}(\mathcal{G})$, where $\boldsymbol{c}$ is a time interval and $\mathcal{G}$ is a TGraph, is defined as:

$$\mathsf{trim}^T_{\boldsymbol{c}}(\mathcal{G}) = (\ TV' = \pi_{v,T \text{ intersect } \boldsymbol{c},a}(\sigma^T_{T \text{ overlaps } \boldsymbol{c}}(TV)),$$
$$TE' = \pi_{e,v_1,v_2,T \text{ intersect } \boldsymbol{c},a}(\sigma^T_{T \text{ overlaps } \boldsymbol{c}}(TE))).$$

trim is essentially temporal selection over TV and TE; its result contains only those nodes and edges whose periods have a non-empty intersection with $\boldsymbol{c}$, with their periods trimmed to be within $\boldsymbol{c}$. This operation can be performed over TV and TE in any order.

Table 3: $\mathbf{map}^T_v(\pi_{type,name}, \mathcal{G})$

TV			
v	**a**	**T**	
v1	type=p,name=Alice	['15/01,'15/07)	
v2	type=p,name=Bob	['15/02,'15/10)	
v3	type=p,name=Cathy	['15/01,'15/10)	
TE			
e	**v1**	**v2**	**a**	**T**
e1	v1	v2	type=co-author,cnt=3	['15/02,'15/06)
e2	v2	v3	type=co-author,cnt=4	['15/07,'15/10)

3.3 Map

To allow manipulation of node and edge attributes we introduce vertex-map and edge-map operators. These operators apply user-defined map functions to attributes in the same spirit as map in functional languages and as relational projection in TRA.

While the map functions are arbitrary user-specified functions, there are some common cases: vertex-map and edge-map may specify the set of properties to project out or retain, it may aggregate (e.g., COUNT) values of a collection property, or unnest a nested value in a property. In other words, mapping is on an entity-by-entity, tuple-by-tuple basis. The time period can be referenced in the mapping function but it cannot be manipulated directly, and so remains unchanged.

Example 3.5. Consider again TGraph $\mathcal{G}$ in our running example from Table 1. vertex-map with a projection of the type and name properties results in a new TGraph, depicted in Table 3. Observe that two tuples with vertex id v2 in the TV relation in the input produce a single tuple over the combined time period due to coalescing.

Definition 3.6 (Vertex-map). The vertex-map operator, denoted $\mathsf{map}^T_v(f_v, \mathcal{G})$, where f_v is a user-defined mapping function that takes a single vertex, with all its properties, as input, and outputs a modified set of vertex properties, is defined as:

$$\mathsf{map}^T_v(f_v, \mathcal{G}) = (TV' = \{(v, a'|T) \mid (v, a|T) \in \mathsf{TV} \wedge a' = f_v(v, a|T)\},$$
$$TE' = TE\ ).$$

In other words, vertex-map is analogous to temporal projection over TV that retains all attributes of TV but changes the nested attribute a. Because vertex time periods can be accessed but cannot be modified, no changes to TE are necessary to enforce referential integrity. The TV′ relation must be coalesced, which is done automatically by the model. The edge-map operator is defined similarly.

3.4 Subgraph

Temporal subgraph matching is a temporal generalization of subgraph matching [36]; it returns a TGraph matching the input navigational graph pattern that may include temporal predicates.

Example 3.7. Consider TGraph $\mathcal{G}$ from our running example (Table 1). Suppose that we are only interested in non-isolated nodes, i.e., only those that have a non-zero degree. Pattern P_1, shown in Figure 1, yields new TGraph $\mathcal{G}$' depicted in Table 4. Observe that

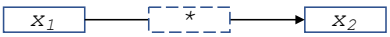

Figure 1: Navigational graph pattern P_1, restricting to non-isolated nodes only.

Table 4: subgraph^T with pattern P_1 returns a graph with no isolated nodes.

TV			
v	**a**		**T**
v1	type=p,name=Alice,school=Drexel		['15/02,'15/06)
v2	type=p,name=Bob		['15/02,'15/05)
v2	type=p,name=Bob,school=CMU		['15/05,'15/10)
v3	type=p,name=Cathy,school=Drexel		['15/07,'15/10)
TE			
e	**v1**	**v2**	**a**	**T**
e1	v1	v2	type=co-author,cnt=3	['15/02,'15/06)
e2	v2	v3	type=co-author,cnt=4	['15/07,'15/10)

while the number of node tuples in the output is the same as in the input, their periods are shorter, and only include time instants when edges are present. For instance, $v1$ period has been reduced from ['15/01,'15/07) to ['15/02,'15/06), since no edge connected to $v1$ exists in either ['15/01,'15/02) or ['15/06,'15/07).

To state this formally, we first modify the definition of a navigational graph pattern [3] to include time:

Definition 3.8 (Temporal Navigational Graph Pattern). A temporal navigational graph pattern (TNGP) is a graph, where V is extended with a set of node variables, E is extended with a set of edge variables and regular path expressions, and property names and values are extended with variables. Variable expressions may have temporal predicates.

A match of TNGP in G is restricted to be isomorphic, i.e., one entity, whether node or edge, cannot match different variables. When no temporal predicates are present in the TNGP, it is semantically evaluated as a regular navigational graph pattern over each snapshot of $\mathcal{G}$.

Definition 3.9 (Temporal subgraph). The temporal subgraph operator, denoted subgraph$^T(P, \mathcal{G})$, where q_v is a set of all node matches of node variables and constants in a TNGP P, and q_e is a set of all edge matches of edge variables and constants in P, is defined as:
subgraph$^T(P, \mathcal{G}) = (q_v(P, \mathcal{G}), q_e(P, \mathcal{G}), L, \rho, \xi^T, \lambda^T)$.

3.5 Aggregation

Temporal aggregation is a generalization of the graph aggregation operation [36] to include temporal predicates and operate over temporal data. It computes the value of a new node property based on information available at the node itself, at the edges associated with the node, and at its neighbors. Aggregation can be used to compute such properties as in-degree of a node, or the set of countries that the friends of v visited in the past year.

Definition 3.10 (Aggregation). Temporal graph aggregation is defined as: $\text{agg}^T(P, G) = (V, E, L \cup M, \rho, \xi^T, \lambda^{T'})$, where P is a TNGP (Definition 3.8) extended with aggregation functions, M is a set of aggregating properties in P, and $\lambda^{T'}$ is augmented with the set of values of node and edge aggregating expressions in P.

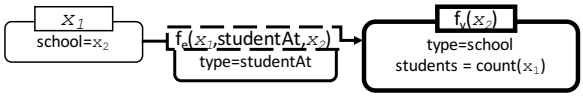

Figure 2: TNGP to create nodes for each value of school.

Informally, graph aggregation produces a new TGraph $\mathcal{G}$' that is isomorphic to $\mathcal{G}$, where each node matching a node in P with an aggregation statement has a new property with the value of that aggregation. The aggregation is over other variables of P, sometimes referred to as *collecting variables*. The usual relational aggregation functions sum, min, max, and count are supported. Additionally, set-based aggregation functions may be included (e.g., place all the group values into a set).

Since aggregation supports recursion, complex whole-graph computation can be expressed. Snapshot analytics [25, 29] can be expressed using an aggregation operation with the snapshot reducibility property. Aggregation can also be used to compute the lengths of *journeys*. A journey is a transitive closure on the graph conditional on the time property of the edges — two nodes are connected by a new edge if there is a non-decreasing time path between them [8].

Example 3.11. Consider TGraph $\mathcal{G}$ in our running example. We can compute the longest journey from each node in $\mathcal{G}$ using aggregation. The longest journey from node $v1$ is of length 2 through node $v2$ to node $v3$, since edge tuple $y2$ starts later than tuple $y1$. However, node $v1$ is unreachable from node $v3$ even if we reverse the edge direction because that requires traveling back in time.

3.6 Node creation

The node creation operator enables the user to analyze an evolving graph at different levels of granularity. This operator comes in two variants — based on node attributes or based on temporal window.

Attribute-based node creation is a temporal generalization of the graph node creation operation [36]. Node creation adds new nodes that represent a matching input pattern. Attribute-based node creation takes in a TNGP extended with Skolem functions, but is otherwise the same as the nontemporal one. Intuitively, the nodes are computed by applying the pattern. Every new node is assigned an identity by a Skolem function. Each new node is connected to the pattern from which it originated with new edges, the identity of which is also assigned by a Skolem function.

Attribute-based node creation allows the user to generate a TGraph in which nodes correspond to disjoint groups of input nodes that agree on the values of all grouping attributes. It can also be used with more complex patterns, such as to generate a new node for each connected component and assign it a size property.

Definition 3.12 (Attribute-based node creation). Attribute-based node creation, denoted $\text{node}_a^T(P^s, \mathcal{G})$ where P^s is a TNGP extended with Skolem functions, is defined as:
$\text{node}_a^T(P^s, \mathcal{G}) = (V \cup M, E \cup N, L, \rho', \xi^{T'}, \lambda^{T'})$, where

Table 5: Attribute-based node creation on property school.

TV			
v		**a**	**T**
v1		type=p,name=Alice,school=Drexel	['15/01,'15/07)
v2		type=p,name=Bob	['15/02,'15/05)
v2		type=p,name=Bob,school=CMU	['15/05,'15/10)
v3		type=p,name=Cathy,school=Drexel	['15/01,'15/10)
Dr		type=school,students=2	['15/01,'15/07)
Dr		type=school,students=1	['15/07,'15/10)
C		type=school,students=1	['15/05,'15/10)
TE			
e	**v1** **v2**	**a**	**T**
e1	v1 v2	type=co-author,cnt=3	['15/02,'15/06)
e2	v2 v3	type=co-author,cnt=4	['15/07,'15/10)
e3	v1 Dr	type=studentAt	['15/01,'15/07)
e4	v3 Dr	type=studentAt	['15/01,'15/10)
e5	v2 C	type=studentAt	['15/05,'15,10)

- M is a set of new nodes created by the Skolem function s_v based on the matches of the variables used as parameters to s_v in P^s,
- N is a set of new edges created by the Skolem function s_e based on the matches of the variables used as parameters to s_e in P^s,
- ρ' is extended to include mappings for new edges in N,
- $\xi^{T'}$ is extended to include mappings for new nodes in M and edges in N,
- $\lambda^{T'}$ is extended to include properties for new nodes in M and new edges in N.

Example 3.13. Consider our running example TGraph $\mathcal{G}$. We can create new summary nodes to represent each school based on the school property of the nodes using the pattern in Figure 2. The result is shown in Table 5, with two new nodes Drexel and CMU and studentAt edges connecting to these nodes. The TNGP can specify aggregation functions to compute properties of the new nodes based on the input pattern; here, count is used to generate the students property.

It is interesting and insightful to analyze an evolving graph at different levels of temporal granularity. The user may want to redefine temporal resolution and look at the graph at that scale, irrespective of whether this resolution is finer or coarser than the natural evolution rate of the graph. For this, we introduce a window-based node creation operator that is similar to the *moving window temporal aggregation* in temporal relational algebra. Our approach is inspired by stream aggregation work of Li et al. [28], adopted to graphs, and by generalized quantifiers of Hsu and Parker [17].

Window-based node creation modifies tuple periods based on consecutive temporal windows from the window specification, such as 2 months or 10 years.

It is useful to be able to quantify required node/edge duration in order to consider it valid. We use *quantifiers* for this purpose. Node and edge quantifiers r_v and r_e are of the form {all|most|at least$n$|exists}, where n is a decimal representing the percentage of the time during which a node or an edge existed, relative to the duration of the window. Quantifiers are useful for observing different kinds of temporal evolution. For example, to observe strong connections over a volatile evolving graph we may include nodes that span the entire window (r_v = all), and edges that span a large portion of the window (r_e = most).

Table 6: $\mathbf{node}^T_w(r_v$ = **always, r_e = **exists**, f_{v_1} = **first(name)**, f_{v_2} = **first(school)**, $\mathcal{G})$**

W	
n	**T**
1	['15/01,'15/06)
2	['15/06,'15/10)

TV		
v	**a**	**T**
v1	type=p,name=Alice,school=Drexel	['15/01,'15/06)
v2	type=p,name=Bob,school=CMU	['15/06,'15/10)
v3	type=p,name=Cathy,school=Drexel	['15/01,'15/10)

TE				
e	**v1**	**v2**	**a**	**T**
e1	v1	v2	type=co-author,cnt=3	['15/05,'15/06)
e2	v2	v3	type=co-author,cnt=4	['15/06,'15/10)

Window specification is of the form n {$unit$|changes}, where n is an integer, and $unit$ is a time unit, e.g., 10 min, 3 years, or any multiple of the usual time units. When the window specification is in the form n changes, it defines the window by the number of changes that occurred in $\mathcal{G}$ (affecting any of its constituent relations). Window boundaries are defined left-to-right — from least to most recent.

Window specification generates a temporal relation W with the schema ($d|T$), where each tuple associates a window number d with its period of validity. Putting it all together:

Definition 3.14 (Window-based node creation). The window-based node creation operator, denoted
$\mathsf{node}^T_w(w, r_v, r_e, f_{v_1}(l_{v_1}), \ldots, f_{v_n}(l_{v_n}), f_{e_1}(l_{e_1}), \ldots, f_{e_m}(l_{e_m}), \mathcal{G})$,
where w is the window specification, W is the relation consisting of periods defined by the specification w, r_v and r_e are node and edge quantifiers, and each $f_{v_j}(l_{v_j})$ ($f_{e_j}(l_{e_j})$) specifies an aggregation function f_{v_j} (resp. f_{e_j}) to be applied to a node property l_{v_j} (resp. edge property l_{e_j}) is defined as:

$$\mathsf{node}^T_w(w, r_v, r_e, f_{v_1}(l_{v1}), \ldots, f_{v_n}(l_{vn}), f_{e_1}(l_{e1}), \ldots, f_{e_m}(l_{em}), \mathcal{G}) =$$
$$(\ TV' = \sigma^T_{r_v}(\mathsf{res}(f_{v_1}(l_{v1}), \ldots, f_{v_n}(l_{vn}), \pi^T_{v,a}(TV \times^T W))),$$
$$TE' = \sigma^T_{r_e}(\mathsf{res}(f_{e_1}(l_{e1}), \ldots, f_{e_m}(l_{em}), \pi^T_{e,v1,v2,a}(TE \times^T W)))).$$

Essentially, we compute a temporal cross product, combining each node and edge with every time window in W that it overlaps, project out the dummy window number, unnest (μ), apply the resolve primitive for cases where more than one identity-equivalent tuple is in the same window, select only those that meet the quantification, and finally re-nest (ν). It does not matter in which order computation is applied, i.e., either TV or TE can be computed first, as long as the foreign key constraint is enforced in the final result.

For the purposes of this operator we expand the list of supported aggregation functions to include temporal first (earliest start time) and last (latest start time).

Example 3.15. Table 6 illustrates window-based node creation by change ($w = 3$ changes) with all quantifier for nodes and exists for edges, and first aggregation function for node and edge properties. $v2$ is present in the result starting at '15/05 because it did not exist for the entirety of the first window. Edge id $e1$ is reduced in duration because before '15/05 $v2$ does not exist, and after '15/06 $v1$ does not exist. On the other hand edge $e2$ is extended in duration to cover the whole second window as both of its end points exist and it existed at some point during the window in the input.

3.7 Set-based operators

We support temporal versions of the three binary set operators intersection ($\cap^{TG}$), union ($\cup^{TG}$), and difference ($\setminus^{TG}$).

The union operator produces a TGraph that contains nodes and edges that exist in either $\mathcal{G}_1$ or $\mathcal{G}_2$. There is a graph-specific difference between TRA union and TGA union related to identity, namely, there is no guarantee that a node or an edge with the same identity and at the same time instant has the same property set in both input graphs. Previously published work either avoided this problem by not using the property model (as long as ids match there is no issue) or side-stepped it by requiring that both input graphs be drawn from the same underlying graph. We aim to provide a more general operation without this restriction. To this end, we require aggregation functions as an additional input to $\cup^{TG}$ to resolve identity-equivalent tuples with overlapping periods.

Definition 3.16 (Union). TGraph union operator, denoted $\cup^{TG}_{f_{v1}(l_{v1}),\ldots,f_{vn}(l_{vn}),f_{e1}(l_{e1}),\ldots,f_{em}(l_{em})}$, where each $f_{vj}(l_{vj})$ (resp. $f_{ej}(l_{ej})$) specifies an aggregation function f_{vj} (resp. f_{ej}) applied to a node property l_{vj} (resp. edge property l_{ej}) is defined as:

$$\mathcal{G}_1 \cup^{TG}_{f_{v1}(l_{v1}),\ldots,f_{vn}(l_{vn}),f_{e1}(l_{e1}),\ldots,f_{em}(l_{em})} \mathcal{G}_2 = (TV' = \mathsf{res}(f_{v1},\ldots,f_{vn}, TV_1 \cup^T TV_2), TE' = \mathsf{res}(f_{e1},\ldots,f_{em}, TE_1 \cup^T TE_2)).$$

We use the TRA's $\cup^T$ operator on both constituent relations of $\mathcal{G}$, followed by applying the resolve primitive to coalesce identity-equivalent tuples. With this approach nodes (resp. edges) can have different property sets while still resulting in a valid output TGraph. Note, however, that it is still necessary for the node (resp. edge) identifiers of the two graphs to be from the same universe.

For the purposes of binary operators we expand the set of aggregation functions to include left, which gives precedence to the value of the property from the left operand, and right, which does the opposite. These functions are useful when one of the two input TGraphs contains the ground truth.

The TGraph intersection operator produces a TGraph of nodes and edges that exist in both $\mathcal{G}_1$ and $\mathcal{G}_2$ and is defined in a similar fashion as the union operator:

Definition 3.17 (Intersection). TGraph intersection operator, denoted $\cap^{TG}_{f_{v1}(l_{v1}),\ldots,f_{vn}(l_{vn}),f_{e1}(l_{e1}),\ldots,f_{em}(l_{em})}$, where each $f_{vj}(l_{vj})$ ($f_{ej}(l_{ej})$) specifies an aggregation function f_{vj} (resp. f_{ej}) applied to a node property l_{vj} (resp. edge property l_{ej}) is defined as:

$$\mathcal{G}_1 \cap^{TG}_{f_{v1}(l_{v1}),\ldots,f_{vn}(l_{vn}),f_{e1}(l_{e1}),\ldots,f_{em}(l_{em})} \mathcal{G}_2 = (TV' = \mathsf{res}(f_{v1},\ldots,f_{vn}, \pi^T_{v,a_1\cup a_2}(TV_1 \bowtie^T_v TV_2)), TE' = \mathsf{res}(f_{e1},\ldots,f_{em}, \pi^T_{e,v_1,v_2,a_1\cup a_2}(TE_1 \bowtie^T_{e,v_1,v_2} TE_2))).$$

Instead of using the TRA set intersection operator we must use an inner join to allow for tuples that are identity-equivalent but have different property sets. If we use set intersection, identity-equivalent tuples that are not value-equivalent would be eliminated. We apply the resolve primitive in the usual manner.

The difference operator produces nodes and edges that exist in $\mathcal{G}_1$ but not in $\mathcal{G}_2$. It does not require aggregation functions for resolution but is otherwise defined similarly:

Definition 3.18 (Difference). TGraph difference is defined as:

$$\mathcal{G}_1 \setminus^{TG} \mathcal{G}_2 = (TV' = \pi^T_{TV_1.v,TV_1.a}(\sigma^T_{TV_2.a \text{ is NULL}}(TV_1 \rtimes^T_v TV_2)), TE' = \pi^T_{TE_1.e,TE_1.v1,TE_1.v2,TE_1.a}(\sigma^T_{TE_2.a \text{ is NULL}}(TE_1 \rtimes^T_e TE_2))).$$

Since we operate based on node (resp. edge) identity rather than the tuple equivalence, instead of the TRA's difference operator we use a left outer join, select only those tuples that exist in $\mathcal{G}_1$ but not in $\mathcal{G}_2$, and project out the extra columns.

3.8 Edge creation

Edge creation is a binary operator and is a temporal generalization of the various non-temporal graph products. The reason we call it edge creation is that the edges produced in the output may not exist in either of the two input graphs. The nodes in the two input graphs are combined, while the edges are computed with a temporal navigational graph pattern extended with namespaces.

Definition 3.19. The edge creation operator $\text{edge}^T_P(\mathcal{G}_1, \mathcal{G}_2)$, where N is a set of edges with an expression containing a Skolem function in a TNGP P over $\mathcal{G}_1$ and $\mathcal{G}_2$ extended with namespaces, and each $f_{vj}(l_j)$ specifies an aggregation function f_{vj} to be applied to a node property l_j, is defined as:

$$\text{edge}^T_{P,f_{v1}(l_1),\ldots,f_{vn}(l_n)}(\mathcal{G}_1, \mathcal{G}_2) = (V', E', L', \rho', \xi^{T'}, \lambda^{T'}),$$

where:

- $V' = \mathcal{G}_1.V \cup \mathcal{G}_2.V$, $E' = N$, $L' = \mathcal{G}_1.L \cup \mathcal{G}_2.L$,
- ρ' is a total function that maps each new edge in N to source and destination nodes in V',
- $\xi^{T'}$ is a union of $\mathcal{G}_1.\xi^T$ and $\mathcal{G}_2.\xi^T$, extended to include mappings to time for new edges in N,
- $\lambda^{T'}(v, l_j) = f_{vj}(\mathcal{G}_1.\lambda^T(v, l_j), \mathcal{G}_2.\lambda^T(v, l_j))$,
- $\lambda^{T'}(e, l_i)$ maps an edge in N and a property label in L based on the aggregate expressions in P.

Similar to node creation, a Skolem function is required to assign identity to new edges. Nodes are generated as in TGA union $\cup^{TG}$. Intuitively, edge creation returns a new TGraph from nodes of $\mathcal{G}_1$ and $\mathcal{G}_2$, connected by edges determined by the input pattern.

Edge creation has several important applications. It can be used to transpose TGraph edges or compute friend-of-friend edges, passing in the same TGraph as both arguments. Since P can be recursive and include predicates over the timestamps, edge^T can create new edges representing journeys. By adding a temporal condition to P, we can obtain journeys similar to time-concurrent paths.

4 FORMAL TGA PROPERTIES

Several properties of temporal languages have been studied: temporal groupedness, temporal semi-completeness, and temporal completeness [5]. We show that the TGraph model is temporally ungrouped but strongly equivalent to the canonical temporally grouped model. We also show that TGA is snapshot reducible with respect to a generalized nontemporal graph query language described in [36]. Additionally, we show that TGA is not temporally semi-complete or temporally complete.

4.1 Temporal Groupedness

Clifford et al. define two basic strategies for adding temporal information into the relational model: temporally ungrouped (tuple timestamping or first-normal-form) and temporally grouped (attribute timestamping or non-first-normal-form) [11]. Temporal relations in TRA, based on the TSQL2 model, are temporally ungrouped [5], since each tuple is associated with its period of validity. In such a model, a change to one attribute results in a new tuple. In a temporally grouped model related facts are grouped and their attributes are associated with their individual periods of validity, with the use of a function. Formally:

Definition 4.1 (Canonical temporally ungrouped relation model). [adapted from [11], pp. 69-70] Let $U_D = \{D_1, \ldots, D_{n_d}\}$ be a set of non-empty value domains, and $\mathbf{D} = \bigcup_{l=1}^{N_d} D_l$ be the set of all values. Let $\Omega^T = \{t_0, \ldots, t_i, \ldots\}$ be a non-empty, at most countably infinite, set of times with total order. Let $U_A = \{A_1, \ldots, A_n\}$ be a set of nontemporal attributes, and T a distinguished time attribute not in U_A. A temporally ungrouped (TU) relation schema R_{TU} is a 3-tuple $R_{TU} =< \mathbf{A}, \mathbf{K}, \mathbf{DOM} >$, where:

- $\mathbf{A} \cup \{T\} (\mathbf{A} \subseteq U_A)$ is the set of attributes of the schema.
- $\mathbf{K} \cup \{T\} (\mathbf{K} \subseteq \mathbf{A})$ is the key of the schema, $\mathbf{K} \cup \{T\} \rightarrow \mathbf{A}$.
- $\mathbf{DOM} : \mathbf{A} \cup \{T\} \rightarrow U_D \cup \{\Omega^T\}$ is a function that assigns domains to attributes in $\mathbf{A}$ and T to Ω^T.

A TU database schema DB_{TU} is a finite set of TU relation schemas. A tuple rt on schema R_{TU} is a function that associates a value for each attribute $A_i \in \mathbf{A}$ in domain $\mathbf{DOM}(A_i)$ and a value in T to Ω^T. A TU relation is a finite set of TU tuples satisfying the key constraint.

A canonical temporally grouped (TG) relation maintains the notion of an object changing over time. As we will show that the TGraph model is isomorphic to TU, we do not include the formal definition of the TG model. Refer to Clifford [11] for the definition and additional discussion. What makes TU relations ungrouped is that there is no mechanism to identify the tuples that correspond to the same entity over time, i.e., there is no unique grouped relation for an ungrouped relation. Without a grouping mechanism the TU model and the TG model are *weakly* equivalent [11].

The TGraph model is ungrouped by the virtue of being expressed over an ungrouped relational model of TRA. However, the TGraph model requires that each node and edge have an id that serves as a key and persists over time, or, in Clifford terminology, it uses constant keys for a group identifier [11]. With this restriction the TGA model is strongly equivalent to the canonical temporally grouped model.

The reasons that we chose an ungrouped model are two-fold: a) it is the de-facto standard in the temporal relational database community after more than two decades of discussion, and b) it is conceptually easier to express temporal predicates over the ungrouped model.

4.2 Temporal Completeness

Böhlen, et al. define two kinds of upwards compatability with respect to a nontemporal model: temporal semi-completeness and temporal completeness [5]. To define both we need to recollect the notion of snapshot reducibility.

Snapshot reducibility states that for every nontemporal query q in language L, there must exist a corresponding temporal query q^t in the temporal language L^t that generalizes q [12]. Note that this definition does not pose any restrictions on the syntax of the temporal query, so it may be expressed quite differently than the nontemporal one. It also does not restrict the temporal language from having other operators with no nontemporal counterparts.

We can show that TGA is snapshot reducible with respect to a particular graph query language.

Theorem 4.2. *TGA satisfies the snapshot reducibility properties below with respect to the generalized graph query language based on the survey by Wood [36]. $snap_p$ is an operator over TGraphs that computes a graph snapshot at point p.*

(1) $\forall p \in \Omega^T (snap_p(map_v^T(f_v, \mathcal{G})) \Leftrightarrow map_v(f_v, snap_p(\mathcal{G})))$

(2) $\forall p \in \Omega^T (snap_p(map_e^T(f_e, \mathcal{G})) \Leftrightarrow map_e(f_e, snap_p(\mathcal{G})))$

(3) $\forall p \in \Omega^T (snap_p(subgraph^T(P, \mathcal{G})) \Leftrightarrow subgraph(P, snap_p(\mathcal{G})))$

(4) $\forall p \in \Omega^T (snap_p(agg^T(P, \mathcal{G})) \Leftrightarrow agg(P, snap_p(\mathcal{G})))$

(5) $\forall p \in \Omega^T (snap_p(node_a^T(P, \mathcal{G})) \Leftrightarrow node(P, snap_p(\mathcal{G})))$

The equivalences hold for arbitrary TGraphs. The only restriction in this theorem is that any pattern P and mapping function f_v/f_e do not refer to the time attribute, in line with snapshot reducibility.

The proof is included in Appendix B.

Extended snapshot-reducibility requires that L^t provide an ability to make explicit references to timestamps alongside nontemporal predicates. TGA is extended snapshot-reducible because, for each operator in TGA that includes patterns or predicates, temporal predicates are explicitly supported, according to Definition 3.8.

As noted, snapshot reducibility places no restrictions on the syntax of the temporal language. To deal with the syntax, Böhlen, et al., define a more restrictive type of upward compatability termed *temporal semi-completeness* [5].

Definition 4.3 (Temporal semi-completeness). [[5], p. 162] Let $M = (DS, L)$ be a nontemporal data model and $M^T = (DS^t, L^t)$ be a valid-time temporal data model. Then data model M^t is temporally semi-complete with respect to M iff:

(1) For every relation $\mathbf{r}$ in DS there exists a temporal relation $\mathbf{r^t}$ in DS^T such that $\mathbf{r} = \tau_t(\mathbf{r^t})$.
(2) For every query q in L, there exists a query q^t in L^t that is snapshot reducible with respect to q.
(3) There exist two text strings S_1 and S_2 such that for all pairs of queries (q, q^t), where q^t is snapshot reducible to q, query q^t is syntactically identical to $S_1 q S_2$.

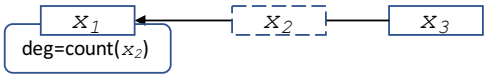

Figure 3: Temporal navigational graph pattern p_1 to compute node degrees.

Temporal semi-completeness provides a kind of upward compatability that simplifies the transition of nontemporal language users to the temporal variant.

The TGraph model meets condition 1, since every snapshot graph can be modeled as a TGraph with a unit interval as its timestamp. The use of duplicate free set-based semantics is not an issue because graph snapshots do not allow duplicates either — every node and edge has an identity. Condition 2 is also satisfied, as we already showed that for every graph operator there is a corresponding TGA operator that is snapshot-reducible.

Since TGA does not specify a syntax, requirement 3 cannot be satisfied. The definitions in Section 3 specify the semantics of each operator, but do not dictate any specific way the operator must be expressed by the user. For instance, we do not specify how the patterns are specified. Thus TGA cannot be considered temporally semi-complete with respect to any graph query language.

A further notion of upward compatability is *temporal completeness*, which requires semi-completeness, as well as an ability to override snapshot reducibility and to refer to original intervals as specified by the user [5]. The last condition is termed *change preservation* [12] and is a property of sequenced semantics. TGA is not temporally complete both because it is not temporally semi-complete and because it uses point-based semantics, thus violating change preservation.

5 USE CASES

An interaction network is one typical kind of an evolving graph. It represents people as nodes, and interactions between them such as messages, conversations, and endorsements, as edges. Information describing people and their interactions is represented by node and edge attributes. One easily accessible interaction network is the wiki-talk dataset [35], containing messaging events among Wikipedia contributors over a 13-year period. Information available about the users includes their username, group membership, and the number of Wikipedia edits they made. Messaging events occur when users post on each other's talk pages.

We now present common analysis tasks and show how they can be expressed in TGA. We are primarily interested in analyses of the *evolution* of the phenomena the graph represents in the form of queries. The examples were selected for their breadth, based on our analysis of related work.

5.1 Node Influence Over Time

In an interaction graph, node centrality is a measure of how important or influential nodes are in the graph. Node importance fluctuates over time. To see whether the wiki-talk graph has high-importance nodes, and how stable node importance is over time during a particular period of interest, we can:

(1) Select a 5-year subset using trim:

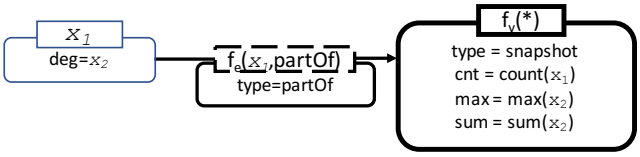

Figure 4: Navigational graph pattern p_2.

$\mathcal{G}_1 = \text{trim}^T_{[2010,2015)}(wikitalk)$

(2) Compute in-degree (prominence) of each node during each time point using aggregation and pattern p_1 depicted in Figure 3: $\mathcal{G}_2 = \text{agg}^T_{p_1}(\mathcal{G}_1)$

(3) Aggregate degree information per node across the timespan of $\mathcal{G}_2$, collecting values into a map using the window-based node creation operator:

$\mathcal{G}_3 = \text{node}^T_w(\text{w} = \text{lifetime}, \text{f}_\text{v} = \{\text{map(deg)}\}, \mathcal{G}_2)$

(4) Transform the attributes of each node to compute the coefficient of variation from the map of degree values, using vertex-map:

$\mathcal{G}_4 = \text{map}^T_v(\text{f}_\text{v} = \text{stdev(deg)/mean(deg)} * 100, \mathcal{G}_3)$

5.2 Graph Centrality over Time

Graph centrality is a popular measure that is used to evaluate how connected or centralized the community is. This measure can be computed by aggregating in-degree values of graph nodes and may change as communication patterns evolve, or as high influencers appear or disappear. In wiki-talk the graph centrality consistently diminishes as the graph grows over time. In sparse interaction graphs there is an additional question of temporal resolution to consider: if two people communicated on May 16, 2010, how long do we consider them to be connected?

This example demonstrates the need to compute graph centrality at every point in its lifetime and to do so at different temporal resolution. For most graph centrality measures, the two-step process involves first calculating some measure, such as in-degree, for each graph node, and then accumulating them into one. To analyze how the graph centrality measure changed over time, we can:

(1) Compute a temporally aggregated view of the graph into windows using the window-based node creation operator with always node and edge quantifiers:

$\mathcal{G}_1 = \text{node}^T_w(\text{w} = 2\text{ mon}, \text{r}_\text{v} = \text{always}, \text{r}_\text{e} = \text{always}, wikitalk)$

(2) Compute in-degree of each node using aggregation:

$\mathcal{G}_2 = \text{agg}^T_{p_1}(\mathcal{G}_1)$

(3) Group all nodes that are present at a given time into a single node with the attribute-based node creation operator, using pattern p_2 in Figure 4. Accumulate max, sum, and count of deg as properties at that node: $\mathcal{G}_3 = \text{node}^T_a(p_2, \mathcal{G}_2)$

(4) Compute centrality at each time point using vertex-map:

$\mathcal{G}_4 = \text{map}^T_v(\text{f}_\text{v} = \text{cent} = (\text{max}*\text{cnt}-\text{sum})/(\text{cnt}^2-3*\text{cnt}+2), \mathcal{G}_3)$

5.3 Spread of Information

We can analyze evolving interaction networks to study how information spreads over time. Suppose each edge has a topic attribute that indicates what each communication between users is about. To see how far and how quickly information spreads, we can compute *journeys*. A journey connects any two nodes by a new edge if there

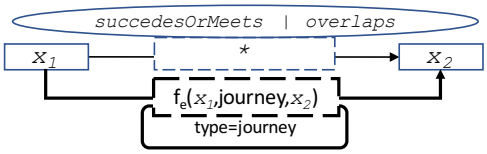

Figure 5: A journey pattern p_7.

is a path between them such that the edge times are non-decreasing. For example, a journey can be over edges that all co-exist or over edges that follow each other in time. In this example, we connect any two nodes that have a path between them on the same topic and assign new edges a length property equal to the sum of the edges in the path. This is an example of an edge creation operation extended for evolving graphs. To see how far information may travel and who the sources are, we can select a subset of the network, consisting only of nodes that originate the longest edges.

(1) Select only the portion of the graph that relates to the topic of interest using the subgraph operator and a pattern with topic = 'Hurricane Sandy' pattern (not depicted):
$\mathcal{G}_1 = \text{subgraph}^T(p_6, wikitalk)$

(2) Connect every pair of nodes that has a path between them with non-decreasing time periods using the edge creation operator using pattern p_7 in Figure 5: $\mathcal{G}_2 = \text{edge}^T_{p_7}(\mathcal{G}_1, \mathcal{G}_1)$

(3) Select the node with the largest number of edges using the vertex-subgraph operator and a pattern with max(sum(x)) expression (not depicted): $\mathcal{G}_3 = \text{subgraph}^T(p_8, \mathcal{G}_2)$

6 RELATED WORK

Conceptual representations. While the temporal models in the relational literature are very mature, the same cannot be said for the evolving graphs literature. Evolving graph models differ in what time representation they adopt (point or interval), 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.

The first mention of evolving graphs that we are aware of is by Harary and Gupta [16] who informally proposed to model graph evolution as a sequence of static graphs. This model has been predominant in the research literature since [14, 20, 21, 27, 32, 33], with various restrictions on the kinds of changes that can take place during graph evolution. For example, Khurana and Deshpande use this model but a node, once removed, cannot reappear [21]. In others [25, 32] there is no notion of time at all, only a sequence of graphs. In [7] and [20] the time series of graphs represent topology only, with no attributes, and only edges can vary with time, while the nodes remain unchanged. Ferreira's model [14] allows both node and edge evolution, but again, restricted to topology.

In contrast to existing work, TGraph assigns periods of validity to nodes, edges and their properties, and can thus capture evolution of graph topology and of node and edge attributes. We use TGraph to supports principled point-based semantics for evolving graphs.

Evolving graph queries. Evolving graph analysis and mining have been receiving increased attention over the past few years [2]. Several researchers have proposed individual queries, or classes of queries, for evolving graphs, but without a unifying framework.

The proposed operators can be divided into temporal, such as those that return a temporal result, and non-temporal. Temporal operators include retrieval of version data for a particular node and edge [22], of journeys [8, 15], subgraph by time or attributes [18, 22], snapshot analytics [22, 25, 29], and computation of time-varying versions of whole-graph analytics like maximal time-connected component [14] and dynamic graph centrality [27].

Non-temporal operators are focused on local and global point queries, including retrieval of specific nodes, edges, or routes at a specific time point, supported in the Historical Graph Store [22] and in Koloniari et al. [23]. George et al. support anomaly detection through a subgraph query to locate specific nodes that have anomalous readings [15]. Khurana and Deshpande provide a snapshot retrieval operation for a specific time point [21], as does the G* system [25]. The main focus in all these approaches is the physical layout of the data and the access methods.

Three systems in the literature focus on systematic support of evolving graphs, all of them non-compositional. ImmortalGraph [29] is a proprietary in-memory execution engine for temporal graph analytics. It does not provide a query language, focusing primarily on efficient physical data layout. G* [25] manages graphs that correspond to periodic snapshots, with a focus on efficient data layout. Time is not an intrinsic part of the system, as it is in TGA, and thus queries with time predicates like node creation are not supported. G* provides graph operators to retrieve vertices and edges from disk, and non-graph operators like aggregate, union, projection, and join. Finally, Historical Graph Store (HGS) [22] is an evolving graph query system that uses the property graph model and supports retrieval tasks along time and entity dimensions through Java and Python APIs. HGS provides a range of operators such as selection, timeslice, nodecompute, and various evolution-centered operators. HGS does not provide formal semantics for any of its operations, focusing on the efficient on-disk representation for retrieval.

TGA is strictly more expressive than the set of operations supported by any of these three systems, is compositional, and is the only evolving graph algebra to provide node and edge creation and support for temporal predicates.

7 CONCLUSION

In this paper we presented TGraph, an extension of the property graph model with tuple timestamping to represent evolution of graph topology, and of vertex and edge attributes. We proposed a compositional algebraic query language called TGA, with point-based semantics. TGA is temporally ungrouped, snapshot reducible, and extended snapshot reducible. We examined several use cases and showed how TGA can be used to express them.

A limitation of TGA that prevents it from achieving temporal completeness is lack of change preservation, to support sequenced semantics. Work on adapting TGA for sequenced semantics is in progress [30].

We developed a prototype implementation of TGraph and TGA on top of Apache Spark, which we will make available to the public shortly. It is in our immediate plans to develop a declarative syntax for TGA, making it accessible to a wider audience of users.

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
[2] C. C. Aggarwal and K. Subbian. Evolutionary network analysis. *ACM Comput. Surv.*, 47(1):10:1–10:36, 2014.
[3] R. Angles et al. Foundations of modern graph query languages. *CoRR*, abs/1610.06264, 2016.
[4] M. H. Böhlen et al. Coalescing in temporal databases. In *VLDB*, 1996.
[5] M. H. Böhlen, C. S. Jensen, and R. T. Snodgrass. Evaluating the Completeness of TSQL2. In *Recent Advances in Temporal Databases*, pages 153–172, 1995.
[6] M. H. Böhlen, C. S. Jensen, and R. T. Snodgrass. Temporal Statement Modifiers. *ACM TODS*, 25(4):407–456, 2000.
[7] K. M. Borgwardt, H. P. Kriegel, and P. Wackersreuther. Pattern mining in frequent dynamic subgraphs. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 818–822, Hong Kong, 2006.
[8] A. Casteigts et al. Time-Varying Graphs and Dynamic Networks. In *ADHOC-NOW*, volume 6811, 2011.
[9] J. Chan, J. Bailey, and C. Leckie. Discovering correlated spatio-temporal changes in evolving graphs. *Knowledge and Information Systems*, 16(1):53–96, 2008.
[10] J. Chomicki. Temporal query languages: A survey. In *ICTL*, 1994.
[11] J. Clifford, A. Croker, and A. Tuzhilin. On Completeness of Historical Relational Query Languages. *ACM TODS*, 19(1):64–116, 1994.
[12] A. Dignös, M. H. Böhlen, and J. Gamper. Temporal Alignment. In *SIGMOD*, 2012.
[13] A. Fard, A. Abdolrashidi, L. Ramaswamy, and J. Miller. Towards Efficient Query Processing on Massive Time-Evolving Graphs. In *Proceedings of the 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 567–574, 2012.
[14] A. Ferreira. Building a reference combinatorial model for MANETs. *IEEE Network*, 18(5):24–29, 2004.
[15] B. George, J. M. Kang, and S. Shekhar. Spatio-Temporal Sensor Graphs (STSG). *Intelligent Data Analysis*, 13(3):457–475, 2009.
[16] F. Harary and G. Gupta. Dynamic graph models. *Mathematical and Computer Modelling*, 25(7), 1997.
[17] P.-y. Hsu and D. S. Parker. Improving SQL with Generalized Quantifiers. In *ICDE*, 1995.
[18] W. Huo and V. J. Tsotras. Efficient Temporal Shortest Path Queries on Evolving Social Graphs. In *SSDBM*, 2014.
[19] C. S. Jensen and R. T. Snodgrass. Temporal data models. In *Encyclopedia of Database Systems*, pages 2952–2957. Springer US, Boston, MA, 2009.
[20] A. Kan et al. A Query Based Approach for Mining Evolving Graphs. In *AusDM 2009*, volume 101, 2009.
[21] U. Khurana and A. Deshpande. Efficient Snapshot Retrieval over Historical Graph Data. In *ICDE*, 2013.
[22] U. Khurana and A. Deshpande. Storing and Analyzing Historical Graph Data at Scale. In *EDBT*, 2016.
[23] G. Koloniari, D. Souravlias, and E. Pitoura. On Graph Deltas for Historical Queries. In *WOSS*, 2012.
[24] K. G. Kulkarni and J. Michels. Temporal features in SQL: 2011. *SIGMOD Record*, 41(3):34–43, 2012.
[25] A. G. Labouseur et al. The G* graph database: efficiently managing large distributed dynamic graphs. *Distrib. and Parallel Databases*, 33(4):479–514, 2014.
[26] M. Lahiri and T. Berger-Wolf. Mining Periodic Behavior in Dynamic Social Networks. In *2008 Eighth IEEE International Conference on Data Mining*, pages 373–382, 2008.
[27] K. Lerman et al. Centrality Metric for Dynamic Networks. In *MLG*, 2010.
[28] J. Li et al. Semantics and evaluation techniques for window aggregates in data streams. In *SIGMOD*, 2005.
[29] Y. Miao et al. ImmortalGraph: A System for Storage and Analysis of Temporal Graphs. *ACM Transactions on Storage*, 11(3):14–34, 2015.
[30] V. Moffitt and J. Stoyanovich. Towards sequenced semantics for evolving graphs. In *EDBT*, 2017.
[31] A. Montanari and J. Chomicki. *Time Domain*, pages 3103–3107. Springer US, Boston, MA, 2009.
[32] C. Ren et al. On Querying Historical Evolving Graph Sequences. *Proceedings of the VLDB Endowment*, 4(11):726–737, 2011.
[33] K. Semertzidis, K. Lillis, and E. Pitoura. TimeReach: Historical Reachability Queries on Evolving Graphs. In *EDBT*, 2015.
[34] K. Sricharan and K. Das. Localizing anomalous changes in time-evolving graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1347–1358, Snowbird, Utah USA, 2014.
[35] J. Sun and J. Kunegis. Wiki-talk datasets, Apr. 2016.
[36] P. T. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, mar 2012.
[37] L. Yang, L. Qi, Y.-p. Zhao, B. Gao, and T.-y. Liu. Link Analysis using Time Series of Web Graphs. In *Proceedings of Conference on Information and Knowledge Management (CIKM)*, pages 1011–1014, Lisboa, Pprtugal, 2007.

A DEFINITION EQUIVALENCE

THEOREM A.1. *Let $\mathcal{TG}$ be the set of valid TGraphs (per Def. 2.1) and let $\mathcal{TG}^{\mathcal{R}}$ be the set of valid TGraph-Relations (per Def 2.2). There is a bijection between $\mathcal{TG}$ and $\mathcal{TG}^{\mathcal{R}}$.*

PROOF. We prove this theorem in two parts. In the first part, we show that, given $\mathcal{G} \in \mathcal{TG}$, we can construct a $\mathcal{G}^{\mathcal{R}} \in \mathcal{TG}^{\mathcal{R}}$ in polynomial time. In the second part, we show that, given $\mathcal{G}^{\mathcal{R}} \in \mathcal{TG}^{\mathcal{R}}$, we can construct a $\mathcal{G} \in \mathcal{TG}$ in polynomial time.

Part 1:

(1) Assume that Ω^T is a set of all time instances in $\mathcal{G}$. For each node $v \in V$ and property in L, we retrieve the set of property values and their periods of validity by applying function λ^T to each point in Ω^T. The input size is $|V|$. The output size is at most $|V| \times |\Omega^T| \times |L|$ if every node exists at every time instant of Ω^T and has every property in L. Recollect that the time domain is of limited precision and that we use point-based semantics, so we can probe every point once. The result is a relation instance $\mathbf{b}$ with schema $(v, l, val \mid T)$, where l is the property name and val is the property value.
(2) Nest l and val into a single nested attribute a. The result is a relation with the same schema as TV and of size at most $|V| \times |\Omega^T|$. Coalesce the result, which can be done in polynomial time [4].
(3) For each edge $e \in E$ retrieve the set of intervals of validity and property sets in the same fashion as in points 1 and 2 above. The result is a relation instance $\mathbf{d}$ with a schema $(e, a \mid T)$, where a is the property relation. As in points 1-2, the input size is $|E|$ and the output size is $|E| \times |T|$.
(4) For each edge $e \in E$ retrieve the source and destination nodes using function ρ. The result is a nontemporal relation instance $\mathbf{f}$ with schema (e, v_1, v_2). The input size and the output size are the same: $|E|$.
(5) $\mathsf{TE} = \mathbf{d} \bowtie^T_e \mathbf{f}$. We join relation instances $\mathbf{d}$ and $\mathbf{f}$ by key, and the size of the result is $|\mathbf{d}|$, as there is a single tuple per edge in $\mathbf{f}$ (edge source and destination points cannot change through time).

The above steps may be simplified if the set of validity periods of the functions ξ^T and λ^T are directly available.

Part 2:

(1) The set of nodes V is a result of nontemporal projection of TV onto key v: $V = \pi_v(\mathsf{TV})$. The input size has an upper bound of $|V| \times c$, where c is the maximum number of changes of any $v \in V$. The output size is $|V|$. Note: this can be expressed solely in TRA as a union of timeslices over temporal projection onto v: $V = \bigcup_{p \in T} \tau_p(\pi^T_v(\mathsf{TV}))$.
(2) The set of edges E is a result of nontemporal projection of TE onto key e: $E = \pi_e(\mathsf{TE})$. The output size is $|E|$.
(3) The set of available properties $L = \pi_l(\mu^T(\mathsf{TV})) \cup \pi_l(\mu^T(\mathsf{TE}))$. The input size is the total number of node tuples, upper-bounded by $|V| \times c$. Unnesting TV has an upper bound of size $|V| \times c \times |L|$ if every node has every property in L. The same applies for TE. The final nontemporal projection leads to all the properties available, at most $|L|$.
(4) Function $\rho(e_1)$ is computed by a key lookup on e_1 in TE, i.e., $\pi^T_{v_1, v_2}(\sigma^T_{e=e_1}(\mathsf{TE}))$. The size of the output is either 0 or 1, since edge end points cannot change with time.

(5) Function ξ^T is computed by a selection query, followed by a slice on TV for nodes (TE for edges resp.): $\xi^T(v_1, t) = \tau_t(\sigma^T_{v=v_1, T=t}(\mathsf{TV}))$, $\xi(e_1, t) = \tau_t(\sigma^T_{e=e_1, T=t}(\mathsf{TE}))$. The size of the output is either 0 or 1.
(6) Function λ^T for nodes is computed by a selection query with an equality predicate on l on TV, followed by a slice: $\lambda^T(v_1, l_i, t) = \tau_t(\sigma^T_{l=l_i}(\mu^T(\sigma^T_{v=v_1, T=t}(\mathsf{TV}))))$. The size of the first selection is at most 1, since only one tuple may exist in TV for a node at any given time (per Def. 2.2, requirement R1). The output of unnesting is of size at most $|L|$ if node v_1 has all the possible properties in L. Finally, the output of the last selection is of size at most 1 if the node v_1 has property l.
(7) Function λ^T for edges is computed by a selection query with a temporal predicate and an equality predicate on l on TE, followed by a slice: $\lambda^T(e_1, l_i, t) = \tau_t(\sigma^T_{l=l_i}(\mu^T(\sigma^T_{e=e_1, T=t}(\mathsf{TE}))))$. The same argument applies as in point 6.

□

B PROOF OF THEOREM 4.2

Proof. To prove this Theorem we consider each equivalence in turn, using either the property graph (Definition 2.1) or the relational definition (Definition 2.2) as convenient.

vertex-map:

$$\mathsf{snap}_p(\mathsf{map}^T_v(f_v, \mathcal{G})) = \mathsf{snap}_p(\mathsf{map}^T_v(f_v, (\mathsf{TV}, \mathsf{TE}))) =$$
$$\mathsf{snap}_p(\ (\{(v, a' \mid T) \mid (v, a \mid T) \in \mathsf{TV} \wedge a' = f_v(v, a \mid T)\}, \mathsf{TE})\ ) =$$
$$(\{(v, a') \mid p \in T \wedge (v, a \mid T) \in \mathsf{TV} \wedge a' = f_v(v, a \mid T)\}, \tau_p(\mathsf{TE}))$$

$$\mathsf{map}_v(f_v, \mathsf{snap}_p(\mathcal{G})) = \mathsf{map}_v(f_v, \mathsf{snap}_p(\ (\mathsf{TV}, \mathsf{TE})\ )) =$$
$$\mathsf{map}_v(f_v, (\tau_p(\mathsf{TV}), \tau_p(\mathsf{TE}))) =$$
$$\mathsf{map}_v(f_v, (\{(v, a) \mid (v, a \mid T) \in \mathsf{TV} \wedge p \in T\}, \tau_p(\mathsf{TE}))) =$$
$$(\{(v, a') \mid (v, a \mid T) \in \mathsf{TV} \wedge p \in T \wedge a' = f_v(v, a)\}, \tau_p(\mathsf{TE}))$$

To show that these two are equivalent, we exploit the commutativity of conjunction to rewrite the $p \in T \wedge (v, a \mid T) \in \mathsf{TV}$ to $(v, a \mid T) \in \mathsf{TV} \wedge p \in T$. Since the theorem restricts f_v from using time, $f_v(v, a \mid T) = f_v(v, a)$. The equivalence follows.

edge-map:

$$\mathsf{snap}_p(\mathsf{map}^T_e(f_e, \mathcal{G})) = \mathsf{snap}_p(\mathsf{map}^T_e(f_e, (\mathsf{TV}, \mathsf{TE}))) =$$
$$\mathsf{snap}_p(\ (\mathsf{TV}, \{(e, v1, v2, a' \mid T) \mid (e, v1, v2, a \mid T) \in \mathsf{TE} \wedge$$
$$a' = f_e(e, v1, v2, a \mid T)\})\ ) =$$
$$(\tau_p(\mathsf{TV}), \{(e, v1, v2, a') \mid p \in T \wedge (e, v1, v2, a \mid T) \in \mathsf{TE} \wedge$$
$$a' = f_e(e, v1, v2, a \mid T)\}, \tau_p(\mathsf{TE}))$$

$$\mathsf{map}_e(f_e, \mathsf{snap}_p(\mathcal{G})) = \mathsf{map}_e(f_e, \mathsf{snap}_p(\ (\mathsf{TV}, \mathsf{TE})\ )) =$$
$$\mathsf{map}_e(f_e, (\tau_p(\mathsf{TV}), \tau_p(\mathsf{TE}))) =$$
$$\mathsf{map}_e(f_e, (\tau_p(\mathsf{TV}), \{(e, v1, v2, a) \mid (e, v1, v2, a \mid T) \in \mathsf{TE} \wedge p \in T\})) =$$
$$(\tau_p(\mathsf{TV}), \{(e, v1, v2, a') \mid (e, v1, v2, a \mid T) \in \mathsf{TE} \wedge p \in T \wedge$$
$$a' = f_e(e, v1, v2, a)\})$$

Just as with vertex-map, we exploit the commutativity of conjunction to rewrite $p \in T \wedge (e, v1, v2, a \mid T) \in \mathsf{TE}$ to $(e, v1, v2, a \mid T) \in \mathsf{TE} \wedge p \in T$. Since the theorem restricts f_e from using time, $f_e(e, v1, v2, a \mid T) = f_e(e, v1, v2, a)$. The equivalence follows.

subgraph:

$$\mathsf{snap}_p(\mathsf{subgraph}^T(P, \mathcal{G})) = \mathsf{snap}_p(\ (q_v(P, \mathcal{G}), q_e(P, \mathcal{G}), L, \rho, \xi^T, \lambda^T))$$

⇓ apply definition of get-snapshot

$$(\{v \mid v \in q_v(P, \mathcal{G}) \wedge \exists t(\xi^T(v, t) \wedge p \in t)\},$$
$$\{e \mid e \in q_e(P, \mathcal{G}) \wedge \exists t(\xi^T(e, t) \wedge p \in t)\},$$
$$L, \rho, \lambda^T([p, p+1)))$$

$$\mathsf{subgraph}(P, \mathsf{snap}_p(\mathcal{G})) = \mathsf{subgraph}(P,$$
$$(\{v \mid v \in V \wedge \exists t(\xi^T(v, t) \wedge p \in t)\},$$
$$\{e \mid e \in E \wedge \exists t(\xi^T(e, t) \wedge p \in t)\},$$
$$L, \rho, \lambda^T([p, p+1))))$$

⇓ apply definition of subgraph

$$(\{v \mid v \in V \wedge \exists t(\xi^T(v, t) \wedge p \in t) \wedge$$
$$v \in q_v(P, (\{v \mid v \in V \wedge \exists t(\xi^T(v, t)\ \wedge p \in t)\},$$
$$\{e \mid e \in E \wedge \exists t(\xi^T(e, t) \wedge p \in t)\}, L, \rho, \lambda^T([p, p+1)))))\},$$
$$\{e \mid e \in E \wedge \exists t(\xi^T(e, t) \wedge p \in t) \wedge$$
$$e \in q_e(P, (\{v \mid v \in V \wedge \exists t(\xi^T(v, t) \wedge p \in t)\},$$
$$\{e \mid e \in E \wedge \exists t(\xi^T(e, t) \wedge p \in t)\}, L, \rho, \lambda^T([p, p+1)))))\},$$
$$L, \rho, \lambda^T([p, p+1)))$$

⇓ substitute graph snapshot with G

$$(\{v \mid v \in V \wedge \exists t(\xi^T(v, t) \wedge p \in t) \wedge v \in q_v(P, G)\},$$
$$\{e \mid e \in E \wedge \exists t(\xi^T(e, t) \wedge p \in t) \wedge e \in q_e(P, G)\},$$
$$L, \rho, \lambda^T([p, p+1)))$$

The difference between the two expansions is only in the application of the q_v and q_e to a TGraph or a snapshot graph G. Per Def. 3.8, when the temporal navigational pattern P is restricted to have no temporal predicates, it is equivalent to a regular navigational graph pattern over each snapshot. Thus the snapshot reducibility property is observed by design.

aggregation: The proof for the aggregation operation is essentially the same as for the subgraph, since the aggregation operator uses a TNGP P^T, which reduces to a nontemporal pattern P when no temporal predicates are present.

node creation: The proof for the node creation operation is essentially the same as for the subgraph, since the node creation operator uses a TNGP P^T, which reduces to a nontemporal pattern P when no temporal predicates are present.

□