Querying Temporal Property Graphs

Landy Andriamampianina, Franck Ravat, Jiefu Song, Nathalie Vallès-Parlangeau

Querying temporal property graphs

Landy Andriamampianina[1,2][0000−0001−9636−7007], Franck Ravat[1][0000−0003−4820−841X], Jiefu Song[1,2][0000−0002−2066−7051], and Nathalie Vallès-Parlangeau[1][0000−0002−4463−5177]

[1] IRIT-CNRS (UMR 5505) - Université Toulouse 1 Capitole (UT1), 2 Rue du Doyen Gabriel Marty F-31042 Toulouse Cedex 09, France
firstname.lastname@irit.fr

[2] Activus Group, 1 Chemin du Pigeonnier de la Cépière, 31100 Toulouse, France
firstname.lastname@activus-group.fr

Abstract. Nowadays, in many real-world problems, objects are characterized by properties and interactions that evolve over time. Several temporal property graph models associated with query languages are proposed in the literature to manage the temporal and interconnectivity features of such problems. However, they are not widely used due to the lack of a conceptual view. To overcome this drawback, we propose user-oriented operators to analyze temporal evolution of property graphs. We also define translation rules between our operators and existing property graph query languages to implement them directly. To illustrate the feasibility of our solution, we present two case studies based on a Neo4j and an OrientDB implementations of our operators and show some real-world querying examples.

Keywords: Operators · Temporal evolution · Implementation · Neo4j · OrientDB.

1 Introduction

Many activities involve people (or objects) that interact in many ways over time [26]. For instance, an infectious disease spreads over time through the contacts among a population. Developing applications to manage these activities requires taking into account both the interconnectivity and the temporal features of the latter. In the context of the Information System, the interconnectivity can be managed by a property graph, which connects entities together through relationships and describes them with attributes [4]. The temporality, however, is not studied in this field. In other words, changes in interconnected data over time (adding, removing and updating entities, relationships as well as their attributes) are neither represented nor exploitable in existing property graph based systems. Our research question is therefore how to model and query interconnected data, taking into account their different types of changes over time.

Temporal management of data has been studied in other domains, such as relational databases [14] and Semantic Web [13]. In light of this work, the graph community has been working on classic graph modelling [7] and graph-oriented

databases [3] that have failed to manage data temporality up to now [18,23]. Some works propose different extensions of classic graph data models to integrate time dimension [27]. In parallel, some works have developed their own management systems for temporal graph data [10,11,15,20]. The overall observation is thus that there is no unified framework for both modelling and querying temporal interconnected data [6,24]. Existing solutions focus mainly on specific implementations and lack of a conceptual view to meet the needs of designers and developers [3,6,24].

The contribution of our work is to provide a conceptual solution (independent of any implementation aspects) for designers and developers to model and query temporal interconnected data. To do so, we propose operators for querying temporal graph data, which are based on a model proposed in our previous work [2]. The remainder of the paper is organized as follows. First, we review the literature on the management of temporal graph data (Section 2). Second, we define our conceptual model (Section 3.1) and operators (Section 3.2). Third, we present translation rules of our solution into technical environments in Section 3.3. Fourth, we evaluate the feasibility of our operators, presented in Section 4. Finally, we conclude the paper.

2 Related work

Temporal evolution of interconnected data shows how entities, relationships and their attributes may evolve in time. This evolution can be categorized into two categories: topological evolution (addition and removal of entities and relationships) and attributes evolution (addition, removal and update of attributes of entities and relationships) [27]. To address the management of such data, we study their modelling and querying in the literature.

Property-graph and RDF data models are commonly used in the context of graph data management [3]. The property graph model allows for representing entities and relationships, using nodes linked by edges, and their attributes, using properties contained in nodes and edges [4]. The RDF model allows for representing the semantic links between data, using nodes linked by edges, but does not allow the nesting of data such as properties contained in nodes and edges [17]. Several graph query languages rely on the previous models such as SPARQL (W3C) [3] for the RDF data model, Cypher (Neo4j) [4], G-Core (LDBC) [5] and PQGL (Oracle) [25] (and so on) for the property graph model. However, they only provide basic graph operations for interconnected data [6,24]. To query temporal aspects of interconnected data, it requires an underlying data model that traces evolution. Although the property graph model incorporates some aspects of data we are interested in, it does not take into account the different types of changes that may occur on interconnected data.

To support temporal evolution, some works propose to extend current graph query languages, based on temporal property graph models, to support basic

[3] `https://www.w3.org/TR/rdf-sparql-query/`
[4] `https://neo4j.com/developer/cypher/querying/`

graph operations and add expressions for time dimension querying [10,11,15]. However, graph query languages are database-dependent but not user-oriented. Implementation details behind graph query languages need to be understood to formulate queries. Therefore, the use of these extended graph query languages can be difficult for designers and developers [9]. Moreover, graph query languages may evolve over time (the addition or removal of functionalities) so they are not very consistent as a basis of a new query language [9]. The work of [20] proposes a graph query language independent of current graph query languages. However, it has the same limited composability as for graph query languages. Composability[5] allows complex queries to be built up from smaller (or simpler ones). This facilitates the understanding of complex queries and the reuse of existing operations [5].

Other works propose an algebra of composable operators for querying temporal property graphs models, in the same idea as the relational algebra [19,22]. They extend existing graph operations [3,6,24], such as the *basic graph pattern matching*[6], to incorporate temporal operators functionally similar to Allen operators [1]. They also introduce novel operators such as the *snapshot* operator, which extracts subgraphs (nodes and edges) that existed at a user-defined time range. They allow operators to be combined for advanced analysis, such as comparing two graphs obtained by the snapshot operator to observe changes between the two. However, the algebra in [19] relies on the sequence of property graph snapshots, a graph-specific adaptation of the temporal data modelling strategy in the relational databases [12]. This approach fails to trace changes at the level of the graph component (node and edge). Therefore, it does not allow direct querying of temporal graph data at the graph component level [16]. Both [19] and [22]'s operators manipulate data-oriented concepts from their underlying models, such as nodes and edges, instead of user-oriented concepts, such as entities and relationships. Moreover, there is no technical environment that implements them directly.

To sum up, existing works fail to either provide a user-oriented solution or support all temporal analysis operations. To do so, we propose a solution to query temporal graph data according to the following user-oriented criteria (i, ii and iii) and analysis criteria (iv and v): (i) to rely on a user-oriented conceptual model, (ii) to be composable to facilitate the formulation of temporal queries and their combination, (iii) to be directly implementable, (iv) to support basic and temporal graph operations, and (v) to support direct querying at all levels of the graph (a component, a set of components and the whole graph). To satisfy the previous criteria, we introduce in the following section our conceptual model and operators for representing and querying temporal interconnected data.

[5] Graphs are input and output of queries [5].

[6] Basic graph pattern matching consists in retrieving subgraphs that match a user-defined pattern (or graph structure).

3 Proposition

3.1 Model

We define a conceptual model of temporal property graphs, on which rely our operators. All the details of this model are available in [2].

Our model provides concepts to represent objects, relationships between objects and their evolution in terms of (i) topology i.e., their addition and removal over time as well as (ii) attributes (or properties) i.e, the addition and removal of new attributes and the change in attribute values. To handle such evolution, the model manages time through the concept of *valid time interval,* which represents when something has occurred or changed in the real world. We denote it $T = [t_s, t_f[$ which indicates a time interval starting from the time instant t_s and extending to but excluding the time instant t_f.

To describe an object, we propose the concept of *temporal entity.* A *temporal entity* e_i has an identifier id and a label l that describes its semantic. At each change of an entity, in terms of topology and attributes, a new state of the entity is created instead of overwriting the old state version to keep track of changes. Therefore, a temporal entity is composed of a non-empty set of states S^{e_i}. Each state $s_j \in S^{e_i}$ describes the characteristics of the entity through a set of attributes A^{s_j}, the related set of attribute values V^{s_j} and a valid time interval T^{s_j} indicating the time during which A^{s_j} and V^{s_j} do not change. The function Σ_e returns for a temporal entity all of its states:

$$\Sigma_e : e_i \rightarrow \{s_1, ..., s_m\}$$

A relationship between two objects does not have an independent existence. Its existence depends on the objects it links. To describe such relationship, we propose the concept of *temporal relationship.* A *temporal relationship* r_i is defined according to the same concepts and functions as a temporal entity. Its particularity is to describe the link between two entity states (s_k, s_j). The function ρ returns, for two entity states s_k and s_j and a relationship label l, a temporal relationship r_i if it exists:

$$\rho : s_k \times s_j \times l \rightarrow r_i$$

Following these definitions, a *Temporal Property Graph* is defined by $TG = \langle E, R \rangle$ where $E = \{e_1, ..., e_g\}$ is a finite set of temporal entities and $R = \{r_1, ..., r_h\}$ is a finite set of temporal relationships. Subsets of entities (or relationships) in the graph can share the same semantic. The function δ returns for a label l, the set of entities E_k (or relationships R_k) having the label l:

$$\delta_e : l \rightarrow E_k \text{ where } E_k \subset E$$
$$\delta_r : l \rightarrow R_k \text{ where } R_k \subset R$$

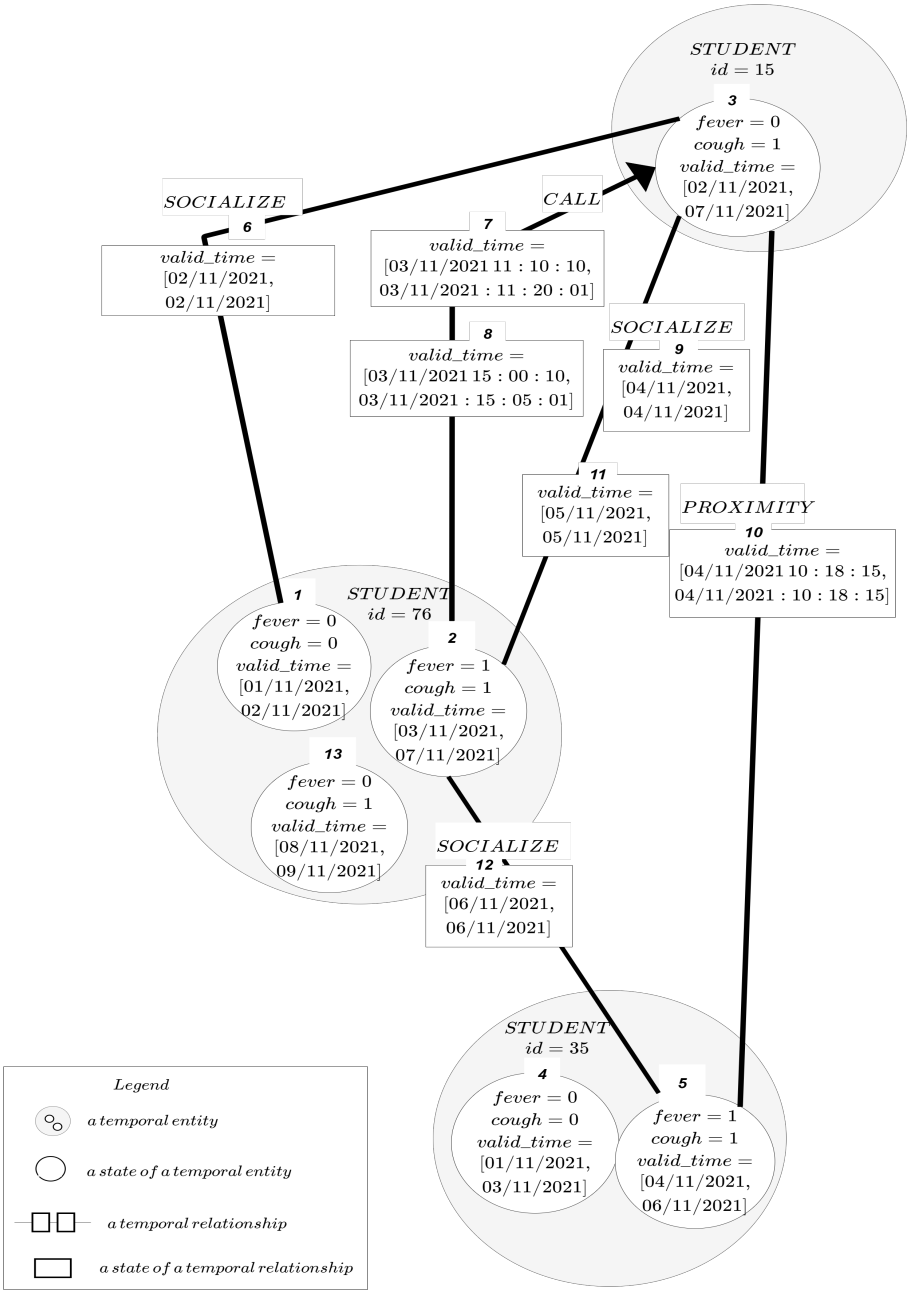

Fig. 1: Graphical notation of our temporal property graph.

Example 1. In this example, we propose a real-world use case of our temporal property graph model to present its graphical notation. A social experiment[7] was designed to study the contagion of flu symptoms of students from MIT. The study includes daily surveys about the flu symptoms of students: cough, fever and so on. Moreover, it includes information about interpersonal relationships (close friends, socialize[8], etc.) and physical information (proximity[9], call, etc.).

We present in Figure 1 a short example of the temporal property graph representation of such dataset. Students are modelled as labelled temporal entities and illustrated by nodes in grey. All descriptive information, such as flu symptoms of students, are modelled through attributes. Students evolve in terms of their attribute values. Indeed, they have different states (illustrated by the white nodes) because their flu symptoms can change over time[10]. For instance, the student identified 35 has two states numbered 4 and 5 because he got fever and cough at state 5 that he did not have at state 4. The interpersonal relationships and physical information between students evolve only according to their topology (addition and removal over time). Each of them is modelled as a temporal relationship between two states of students and illustrated by a black edge. Each temporal relationship is labelled with its semantic (socialize, proximity or call etc.). It can have several states (illustrated by white rectangles) as they occur several times. For instance, the student 76 calls two times the student 15 that is why the relationship $CALL$ has two states numbered 7 and 8.

3.2 Operators

In this section, our objective is to propose user-oriented operators. Instead of proposing a set of operations with a technical view (such as projection), we propose two emblematic operators for temporal graph analysis: (i) extraction of a subgraph satisfying selection and temporal criteria and (ii) extraction of a subgraph satisfying pattern criteria.

Our proposed operators rely on the previous model of temporal property graph (Section 3.1). They return a temporal subgraph TG_{output} whose entities and relationships are subsets of an input temporal property graph TG_{input}.

In the context of temporal interconnected data, users may reason over time dimension to express their analyses [26]. To do so, we propose the $matching_{predicate}$ operator for querying temporal property graphs according to conditions on attributes and time.

[7] available on the Reality Commons website `http://realitycommons.media.mit.edu/socialevolution.html`

[8] participate at least one common activity

[9] Bluetooth signal sent from whose mobile phone and received by whose mobile phone and time, indicating the sender's mobile phone was within 10 meters of the receiver's mobile phone at the time of the record.

[10] The value of a symptom attribute equals 0 if the student does not have the symptom and 1 if the student has the symptom.

Table 1: Predicate matching operator.

Operator:	$matching_{predicate}(TG_{input}, \pi)$
Input:	an input graph TG_{input}; a set of user-defined predicates π
Output:	a subgraph TG_{output}
Actions:	1. Extract all the labels in π 2. $TG_{output} \leftarrow \langle \emptyset, \emptyset \rangle$ 3. For each $l \in \pi$ 4. Extract the set of entities or relationships labelled l from TG_{input} using $\delta_e(l)$ or $\delta_r(l)$ 5. For each $e_h \in \delta_e(l)$ or $r_h \in \delta_r(l)$ 6. Get the states of e_h or r_h using $\Sigma_e(e_h)$ or $\Sigma_r(r_h)$ 7. For each $s_i \in \Sigma_e(e_h)$ or $s_i \in \Sigma_r(r_h)$ linking (s_j, s_k) 8. If s_i satisfies all conditions in π 9. $TG_{output} \leftarrow TG_{output} \cup \langle \{s_i\}, \emptyset \rangle$ 10. or $TG_{output} \leftarrow TG_{output} \cup \langle \{s_j, s_k\}, \{s_i\} \rangle$ 11. End If 12. End For 13. End For 14. End For 15. Return TG_{output}.

Definition 1. *The $matching_{predicate}$ operator is used to extract the subgraph TG_{output} from an input graph TG_{input} according to a user-defined set of predicates on each element of entity and relationship sets of the input graph. Within a predicate, the user has access to an entity or relationship label l, attributes, valid time intervals and can express a logical condition. The algorithm of the execution of time matching operator is presented in Table 1.*

$$matching_{predicate} : TG_{input} \times \pi \rightarrow TG_{output}$$

where $\pi = \{p_1 \, \beta \, p_2 ... p_{n-1} \, \beta \, p_n\}$ is a set of user-defined predicates combined with connective operators β such as AND or OR (etc.).

Each predicate p_i equals $l.a\theta w$ or $l.T\theta w$ where l is an entity or relationship label, a is an attribute, T is a valid time interval, w is a user defined value and θ is a comparison operator.

In the case of an attribute predicate $p_i = l.a\theta w$, θ could be, for instance $=$, $<$ or $>$ (etc.) to express that the attribute a satisfies a given value or range. In the case of a time predicate $p_i = l.T\theta w$, θ is a temporal operator such as an Allen operator presented in Table 2.

Contrary to the filter operation for static property graphs [3,6,24], the proposed $matching_{predicate}$ operator allows filtering graph data not only according to predicates on attributes but also on valid time intervals. As we can see in Table 1, the $matching_{predicate}$ operator extracts only entities and relationships that satisfy the set of predicates at the level of states.

Table 2: Allen operators. $T = [t_s, t_f[$ is a valid time interval. $T_u = [x, y[$ is a user-defined time interval where variables x and y are time instants.

Operators	**Description**	**Relations between time instants**
$T < T_u$	T precedes T_u	$t_f < x$
$T_u > T$	T_u is preceded by T	$x > t_f$
$T\, m\, T_u$	T meets T_u	$t_f = x$
$T \circ T_u$	T overlaps T_u	$max(t_s, x) < min(t_f, y)$
$T\, s\, T_u$	T starts T_u	$t_s = x$
$T\, d\, T_u$	T during T_u	$(t_s > x)$ AND $(t_f < y)$
$T\, f\, T_u$	T finishes T_u	$t_f = y$
$T = T_u$	T equals T_u	$(t_s = x)$ AND $(t_f = y)$

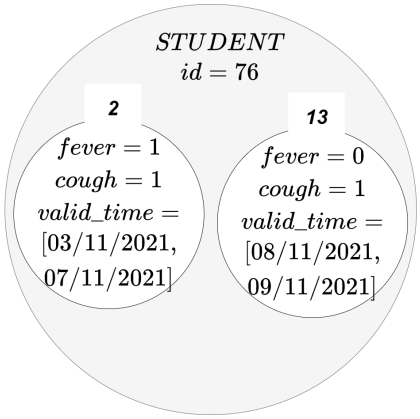

Fig. 2: Result of example 2.

Example 2. Following Example 1, we want to know how did evolve the symptoms of the student 76 since 03/11/2021. To do so, we apply $TG_1 = matching_{predicate}(TG, \{(STUDENT.T \circ [03/11/2021, +\infty[)\, AND\, (STUDENT.id = 76)\})$ to the temporal graph TG in Figure 1. We use the temporal operator "overlaps" denoted $\circ$ in Table 2 to express the time predicate. The previous operation extracts the states of the student identified 76 with a valid time interval overlapping $[03/11/2021, +\infty[$. We obtain the subgraph TG_1 illustrated in Figure 2. We observe that student 76 has two states, meaning that he experienced a change in its flu symptoms.

A fundamental operation of graph analytics is to explore subgraphs that match a user-defined graph pattern. To do so, we propose the following operator.

Definition 2. *A $matching_{pattern}$ operator returns for an input graph TG_{input}, an output subgraph TG_{output} matching a user-defined graph pattern defined by a starting entity label l_{E_i}, an ending entity label l_{E_k}, and a relationship label l_{R_j}*

Table 3: Graph pattern matching operator.

Operator:	$matching_{pattern}(TG_{input}, (l_{E_i}, l_{R_j}, l_{E_k}))$
Input:	two entity labels l_{E_i} and l_{E_k}; a relationship label l_{R_j}
Output:	a subgraph TG_{output}
Actions:	1. $TG_{output} \leftarrow \langle \emptyset, \emptyset \rangle$ 2. Extract the set of entities labelled l_{E_i} and l_{E_k} from TG_{input} using $\delta_e(l_{E_i})$ and $\delta_e(l_{E_k})$ 3. For each $e_h \in \delta_e(l_{E_i})$ 4. Get the states of e_h using $\Sigma_e(e_h)$ 5. For each $e_f \in \delta_e(l_{E_k})$ 6. Get the states of e_f using $\Sigma_e(e_f)$ 7. For each $s_i \in \Sigma_e(e_h)$ 8. While there exists an entity state $s_j \in \Sigma_e(e_f)$ such that $\rho(s_i, s_j, l_{R_j}) \neq \emptyset$ 9. $TG_{output} \leftarrow TG_{output} \cup \langle \{s_i, s_j\}, \{\rho(s_i, s_j, l_{R_j})\} \rangle$ 10. End while 11. End For 12. End For 13. End For 14. Return TG_{output}

to be traversed between the two entity labels. The algorithm of the execution of graph pattern matching operator is presented in Table 3.

$$matching_{pattern} : TG_{input} \times (l_{E_i} \times l_{R_j} \times l_{E_k}) \rightarrow TG_{output}$$

Contrary to the graph pattern matching operation for static property graphs [3,6,24], the proposed $matching_{pattern}$ operator allows extracting subgraph structures with time dependent information. Indeed, it extracts subgraphs having the same pattern as the user-defined graph pattern at the level of states (Table 3). It excludes from the result set the states of entities and relationships that are not labelled with the labels specified in the user-defined graph pattern and that do not verify the existence of relationship states between each pair of entity states.

Example 3. Following Example 1, we want to know if students that had fever socialized since 03/11/2021 with students that did not have fever. To do so, we apply to the temporal graph TG in Figure 1 $TG_2 = matching_{predicate}(TG, \{(STUDENT_1.fever = 0)\, AND\, (STUDENT_2.fever = 1)\, AND\, (SOCIALIZE.T \circ [03/11/2021, +\infty[)\})$ to select entities and relationships satisfying our attribute and time conditions. Then, we apply $TG_3 = matching_{pattern}(TG_2, (STUDENT_1, SOCIALIZE, STUDENT_2))$ on TG_2 to obtain the relationships between the selected entities. We obtain the subgraph TG_3 in Figure 3. We observe that the relationship has two states (numbered

9 and 11) meaning that the relationship has changed. Here, as there is no attribute in the relationship, we only have information about its evolution in terms of topology, i.e. the time of its occurrence (when it is added and removed).

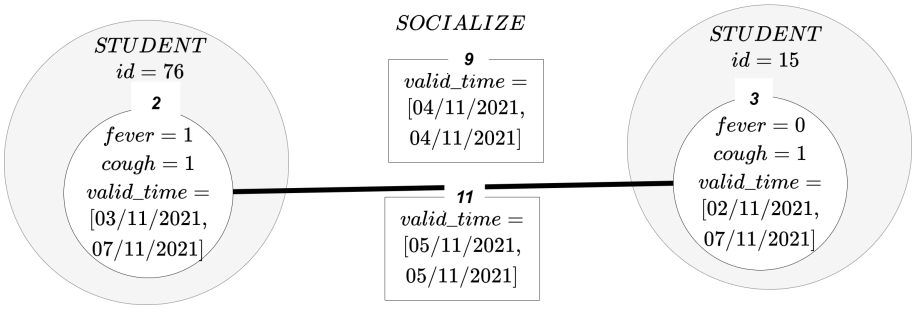

Fig. 3: Result of example 3.

3.3 Mapping from temporal graph operators to a property graph operators

In this section, our objective is to define translation rules of our operators into existing property graph operations to enable a direct implementation in property-graph based systems.

As a preliminary step, we have to map our conceptual temporal property graph TG into a logical property graph PG [4]. A property graph PG is composed of nodes and edges. Each edge is associated to a pair of nodes. Each node (or edge) can be associated with a set of labels and can contain properties. Each property is a key-value pair. For each state s of an entity e in TG, a node is created in PG with a label corresponding to the label of e and a set of properties corresponding to the identifier of e, the attributes of s, the start and end instants of the valid time interval of s. An entity in the property graph corresponds therefore to a set of nodes having the same identifier. For each state s of a relationship r in TG, an edge is created in PG by connecting the two nodes corresponding to two states that r links, with a label corresponding to the label of r and a set of properties corresponding to the attributes of s, the start and end instants of the valid time interval of s. A relationship in the property graph corresponds therefore to a set of edges linking the same pair of nodes. As a result, we obtain the translation rules in Table 4 of our temporal property graph into the property graph to implement the model in any graph oriented NoSQL database compatible with the property graph.

In Table 6, we define translation rules of our proposed operators in existing property graph languages. The algorithms of our proposed operators (Tables 1

Table 4: Translation rules of our conceptual model into the logical model of property graph, the property graph in Neo4j and the property graph in OrientDB. **startvalidtime and endvalidtime.* ***with different valid time intervals.*

Temporal property graph	**Property graph**	**Neo4j**	**OrientDB**
a state of a temporal entity s_j	a node	a node	a vertex
a state of a temporal relationship s_b	an edge	an edge	an edge
a valid time interval of an entity state T^{s_j}	two properties*	two properties	two properties
a valid time interval of a relationship state T^{s_b}	two properties*	two properties	two properties
a temporal entity e_i	a set of nodes**	a set of nodes	a set of vertices
a temporal relationship r_i	a set of edges**	a set of edges	a set of edges
a label of a temporal entity l	a label	a label	a vertex class
a label of a temporal relationship l	a label	a type	an edge class
an attribute of a temporal entity $a_q^{e_i}$	a property	a property	a property
an attribute of a temporal relationship $a_d^{r_i}$	a property	a property	a property

Table 5: Graph operations to query property graphs. *Bgpm = Basic graph pattern matching.*

Operator	Description	Notation
Bgpm	Extracts subgraphs from a property graph G according to pattern P	$\tau(G, P)$
Projection	Returns a subset of the output variables O of the result of a bgpm	$\mu(G, O)$
Filter	Extracts subgraphs from G satisfying user-defined conditions C	$\sigma(G, C)$

and 3) define the scope of the sequence of basic operations in existing property graph query languages (Table 5). A $matching_{predicate}(TG_{input}, \pi)$ operation is translatable into property graph query languages through the execution of the filter operation $\sigma(TG_{input}, \pi)$ to extract the subgraphs G_1 from the temporal property graph TG_{input} satisfying user-defined predicates π. Then, the projection operation $\mu(G_1, G_1)$ returns the graph resulting from the previous operation. A $matching_{pattern}(TG_{input}, (l_{E_i}, L_{R_j}, l_{E_k}))$ operation is translatable into property graph query languages through the execution of the basic pattern matching operation $\tau(TG_{input}, (l_{E_i}, L_{R_j}, l_{E_k}))$ to extract subgraphs G_1 with the pattern $(l_{E_i}, L_{R_j}, l_{E_k})$. Then, it uses the μ operator as in the chain of operations of $matching_{predicate}$ to get the result set.

In some environments, existing property graph query languages include sufficient functionalities to translate easily our proposed operators. In other environments, our proposed operators could not be easily implementable via the property graph query language. Due to the rich functionalities of the query languages of Neo4j and OrientDB graph databases, the mapping of our operators is simple.

Table 6: Translation of our temporal graph operators into property graph query languages. $l.p_x$ and $l.p_y$ are user-defined predicates in π. *TG = Temporal Property Graph, PG = Property Graph, QL = Query Language.*

TG Operators	$matching_{predicate}$ (TG_{input}, π)	$matching_{pattern}$ $(TG_{input}, (l_{E_i}, l_{R_j}, l_{E_k}))$
PG Operators	1. $G_1 = \sigma(TG_{input}, \pi)$ 2. $G_2 = \mu(G_1, G_1)$	$G_1 = \tau(TG_{input}, (l_{E_i}, l_{R_j}, l_{E_k}))$ $G_2 = \mu(G_1, G_1)$
Neo4j QL	$MATCH\,(a : l)$ $WHERE\,l.p_x\,AND\,l.p_y$ $RETURN\,a$	$MATCH\,(a : l_{E_i}) - (b : l_{R_j})$ $-(c : l_{E_k})$ $RETURN\,a, b, c$
OrientBD QL	$MATCH\{class : l, as : a,$ $where : p_x AND p_y\}$ $RETURN *$	$MATCH\{class : l_{E_i}\}.outE(l_{R_j})$ $.inV(l_{E_k})$ $RETURN *$

4 Experimental evaluation

We run an experiment with the objective of evaluating the feasibility of our solution (our model and operators) through their implementation in technical environments.

4.1 Protocol

To validate the feasibility of our solution, we implement two datasets in two different graph database systems.

We used Neo4j and OrientDB graph database systems. More precisely, we used the following hardware configuration for the experiment: PowerEdge R630, 16 CPUs x Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40Ghz, 63.91 GB. Two virtual machines are installed on this hardware. Each virtual machine has 6 GB in terms of RAM and 100 GB in terms of disk size. On each of the two virtual machines, we installed respectively a graph database compatible with the property graph model: (i) Neo4j (version 4.1.3) and (ii) OrientDB (version 3.0.4). To avoid any bias in the disk management, we did not use any customized optimization techniques, but relied on default tuning of Neo4j and OrientDB.

To avoid any bias in datasets, we need to include both real and benchmark datasets. The first one we used is the Social experiment dataset presented in Example 1. The second one is a dataset generated from a reference benchmark available online, namely TPC-DS benchmark[11]. We transformed both datasets into the temporal property graph. Then, we stored the Social

[11] `http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf`

Table 7: Characteristics of datasets. *Y= Yes, N= No, AV = Attribute Value, AS = Attribute Set, T = Topology.*

Implementation	**Social experiment**	**TPC-DS**
Number of nodes	33 934	2 348 965
Number of edges	2 168 270	41 898 261
Size in GB	0.3	17
Evolution types of entities	AV,T	AV,AS,T
Evolution types of relationships	T	AV,AS,T

experiment dataset in Neo4j and the TPC-DS dataset in OrientDB by applying the mapping rules in Table 4. As a result, we obtained two implementations having different volumes (from 0.3 GB to 17 GB), different temporal evolutions (many nodes in one dataset and many edges in another) and different domains (social networks and a retail company). Details of both implementations are presented in Table 7 and available on the website `https://gitlab.com/2573869/queryingtemporalpropertygraphs`.

Finally, we need temporal analyses adapted for each dataset. The objective is to express them into our operators and then translate them into the graph query languages of the graph database systems we used.

4.2 A Neo4j-based implementation

We query the Social Experiment dataset in Neo4j to make temporal analyses based on our operators. The first query concerns the analysis of the evolution of an entity during a period. The second query concerns the analysis of the evolution of several entities at a time point. In the following, we express the previous analyses using our operators and translating them into Cypher, the Neo4j's query language.

Query 1 : *How did evolve the symptoms of student 76 during the period from 19/01/2009 to 23/01/2009 ?* This query corresponds to the following operation in our solution:

- $TG_1 = matching_{predicate}(TG_{input}, \{(STUDENT.id = 76)\ AND\ (STUDENT.Td[19/01/2009, 23/01/2009])\}$

  The above query is translated (Table 6) into Cypher as:

```
MATCH (s:Student)
WHERE s.id="76"
AND date(s.startvalidtime) > date("2009-01-19")
AND date(s.endvalidtime) < date("2009-01-23")
RETURN s
```

Query 2 : *Are there students that had fever and socialized at the day 05/03/2009 with other students that did not have fever ?* This query corresponds to the following sequence of operations in our solution:

- $TG_1 = matching_{predicate}(TG_{input}, \{(STUDENT_1.fever = 1)$
  $AND\,(STUDENT_2.fever = 0)$
  $AND\,(SOCIALIZE.T \circ [05/03/2009, 06/03/2009[)\})$
- $TG_2 = matching_{pattern}(TG_1, (STUDENT_1, SOCIALIZE, STUDENT_2))$

The above query is translated (Table 6) into Cypher as:

```
MATCH p=(s1:Student)-[r:Socialize]-(s2:Student)
WHERE  s1.fever="1"
AND s2.fever ="0"
AND date(r.startvalidtime) < date("2009-03-06")
AND date(r.endvalidtime) >= date("2009-03-05")
RETURN p
```

4.3 An OrientDB-based implementation

We query the TPC-DS dataset in OrientDB to conduct a temporal analysis based on our operators. The query concerns the analysis of the evolution of a set of entities during a period. In the following, we express the previous query using our operators and translating them into OrientDB query language.

Query 3 : *Find the stores that proposed a promotion at the month 03/1996 ?* This query corresponds to the following sequence of operations in our solution:

- $TG_1 = matching_{predicate}(TG_{input}, \{(Promotion.T \circ ([03/1996, 04/1996[)\})$
- $TG_2 = matching_{pattern}(TG_1, (Store, SS\_Promotion, Promotion))$

The above query is translated (Table 6) into OrientDB query language as:

```
MATCH {class: Store, as: s}.outE("SS_Promotion").inV("Promotion")
{as: p, where: (start_valid_time.asDate().format("yyyy-MM")
< date("1996-04", "yyyy-MM").format("yyyy-MM")
AND end_valid_time.asDate().format("yyyy-MM")
>= date("1996-03", "yyyy-MM").format("yyyy-MM"))}
RETURN *
```

5 Conclusion and future work

In this paper, we proposed two conceptual operators to help designers and developers to query temporal property graphs. They combine several advantages compared to current graph query languages and operators. First, they support classic and temporal graph operations. Second, they allow for a direct querying at different levels of the graph and at different time granularity. Third, they

manipulate user-oriented concepts instead of database-oriented concepts to facilitate the formulation of queries. Finally, they are directly implementable into existing property graph database systems. To validate their feasibility, we examined use cases based on real and benchmark datasets. We showed how our operators can be used to express real-world analysis cases, and how they can be easily translated into queries in Neo4j and OrientDB using our translation rules.

In our future work, we will have several research directions. First, regarding our proposition, we will make a theoretical study to prove the completeness of our operators and add new operators if necessary. Second, regarding the evaluation of our solution, we are currently working on the performance evaluation of our operators. This evaluation will include several experiments based on several alternative implementations of graph database management systems and graph query languages, and based on different datasets and query types. Then, we will evaluate the query performance via experiments. To do so, we will study different optimization techniques, and propose several scale factors to test scalability. The objective is to propose guidelines for the implementation choice according to the analytical needs. Finally, we identified a possible extension of our solution in other domains, such as the Semantic Web domain [8,21]. We will verify its implementation in new environments and make a performance comparison with classic graph database systems such as the ones used in our paper.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM **26**(11), 832–843 (1983). https://doi.org/10.1145/182.358434
2. Andriamampianina, L., Ravat, F., Song, J., Vallès-Parlangeau, N.: Towards an Efficient Approach to Manage Graph Data Evolution: Conceptual Modelling and Experimental Assessments. In: Cherfi, S., Perini, A., Nurcan, S. (eds.) Research Challenges in Information Science, vol. 415, pp. 471–488. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-75018-3$_3$1
3. Angles, R.: A Comparison of Current Graph Database Models. In: 2012 IEEE 28th International Conference on Data Engineering Workshops. pp. 171–177. IEEE, Arlington, VA, USA (2012). https://doi.org/10.1109/ICDEW.2012.31
4. Angles, R.: The Property Graph Database Model. In: AMW (2018)
5. Angles, R., Arenas, M., Barcelo, P., Boncz, P., Fletcher, G., Gutierrez, C., Lindaaker, T., Paradies, M., Plantikow, S., Sequeda, J., van Rest, O., Voigt, H.: G-CORE: A Core for Future Graph Query Languages. In: Proceedings of the 2018 International Conference on Management of Data. pp. 1421–1432. ACM, Houston TX USA (2018). https://doi.org/10.1145/3183713.3190654
6. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., Vrgoč, D.: Foundations of Modern Query Languages for Graph Databases. ACM Computing Surveys **50**(5), 68:1–68:40 (2017). https://doi.org/10.1145/3104031
7. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Computing Surveys **40**(1), 1–39 (Feb 2008). https://doi.org/10.1145/1322432.1322433
8. Batsakis, S., Petrakis, E.G., Tachmazidis, I., Antoniou, G.: Temporal representation and reasoning in owl 2. Semantic Web **8**(6), 981–1000 (2017)
9. Bloesch, A., Halpin, T.: ConQuer: A Conceptual Query Language. (Jan 1996)

10. Byun, J., Woo, S., Kim, D.: ChronoGraph: Enabling Temporal Graph Traversals for Efficient Information Diffusion Analysis over Time. IEEE Transactions on Knowledge and Data Engineering **32**(3), 424–437 (2020). https://doi.org/10.1109/TKDE.2019.2891565
11. Debrouvier, A., Parodi, E., Perazzo, M., Soliani, V., Vaisman, A.: A model and query language for temporal graph databases. The VLDB Journal (2021). https://doi.org/10.1007/s00778-021-00675-4
12. Dey, D., Barron, T.M., Storey, V.C.: A Complete Temporal Relational Algebra. VLDB Journal **5**, 5–167 (1996)
13. Gutierrez, C., Hurtado, C., Vaisman, A.: Introducing Time into RDF. IEEE Transactions on Knowledge and Data Engineering **19**(2), 207–218 (Feb 2007). https://doi.org/10.1109/TKDE.2007.34
14. Johnston, T., Weis, R.: A Brief History of Temporal Data Management. In: Managing Time in Relational Databases, pp. 11–25. Elsevier (2010). https://doi.org/10.1016/B978-0-12-375041-9.00001-7
15. Khurana, U., Deshpande, A.: Storing and Analyzing Historical Graph Data at Scale. In: EDBT (2016)
16. Kosmatopoulos, A., Gounaris, A., Tsichlas, K.: Hinode: implementing a vertex-centric modelling approach to maintaining historical graph data. Computing **101**(12), 1885–1908 (Dec 2019). https://doi.org/10.1007/s00607-019-00715-6
17. Lassila, O., Swick, R.R.: Resource description framework (RDF) model and syntax specification (1998)
18. Lazarevic, L.: Keeping track of graph changes using temporal versioning (2019)
19. Moffitt, V.Z., Stoyanovich, J.: Temporal graph algebra. In: Proceedings of The 16th International Symposium on Database Programming Languages. pp. 1–12. DBPL '17, Association for Computing Machinery, Munich, Germany (2017). https://doi.org/10.1145/3122831.3122838
20. Ramesh, S., Baranawal, A., Simmhan, Y.: A Distributed Path Query Engine for Temporal Property Graphs. In: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID). pp. 499–508. IEEE, Melbourne, Australia (2020). https://doi.org/10.1109/CCGrid49817.2020.00-43
21. Ravat, F., Song, J., Teste, O., Trojahn, C.: Efficient querying of multidimensional rdf data with aggregates: Comparing nosql, rdf and relational data stores. International Journal of Information Management **54**, 102089 (2020)
22. Rost, C., Gomez, K., Täschner, M., Fritzsche, P., Schons, L., Christ, L., Adameit, T., Junghanns, M., Rahm, E.: Distributed temporal graph analytics with GRADOOP. The VLDB Journal (2021). https://doi.org/10.1007/s00778-021-00667-4
23. Semertzidis, K., Pitoura, E.: Time Traveling in Graphs using a Graph Database. In: EDBT/ICDT Workshops (2016)
24. Sharma, C., Sinha, R., Johnson, K.: Practical and comprehensive formalisms for modeling contemporary graph query languages. Info. Systems p. 101816 (2021)
25. Van Rest, O., Hong, S., Kim, J., Meng, X., Chafi, H.: PGQL: a property graph query language. In: Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems - GRADES '16. pp. 1–6. ACM Press, Redwood Shores, California (2016). https://doi.org/10.1145/2960414.2960421
26. Wang, Y., Yuan, Y., Ma, Y., Wang, G.: Time-dependent graphs: Definitions, applications, and algorithms. Data Science and Engineering **4**(4), 352–366 (2019)
27. Zaki, A., Attia, M., Hegazy, D., Amin, S.: Comprehensive Survey on Dynamic Graph Models. International Journal of Advanced Computer Science and Applications **7**(2), 573–582 (2016). https://doi.org/10.14569/IJACSA.2016.070273