Graph data temporal evolutions: From conceptual modelling to implementation

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

Graph data temporal evolutions: from conceptual modelling to implementation

Landy Andriamampianina[a,b,*], Franck Ravat[a], Jiefu Song[b], Nathalie Vallès-Parlangeau[a]

[a] *IRIT-CNRS (UMR 5505) - Université Toulouse 1 Capitole (UT1), 2 Rue du Doyen Gabriel Marty F-31042 Toulouse Cedex 09 - France*
[b] *Activus Group, 1 Chemin du Pigeonnier de la Cépière, 31100 Toulouse - France*

Abstract

Graph data management systems are designed for managing highly interconnected data. However, most of the existing work on the topic does not take into account the temporal dimension of such data, even though they may change over time: new interconnections, new internal characteristics of data (etc.). For decision makers, these data changes provide additional insights to explain the underlying behaviour of a business domain. The objective of this paper is to propose a complete solution to manage temporal interconnected data. To do so, we propose a new conceptual model of *temporal graphs*. It has the advantage of being generic as it captures the different kinds of changes that may occur in interconnected data. We define a set of translation rules to convert our conceptual model into the logical property graph. Based on the translation rules, we implement several temporal graphs according to benchmark and real-world datasets in the Neo4j data store. These implementations allow us to carry out a comprehensive study of the feasibility and usability (through business analyses), the efficiency (saving up to 99% query execution times comparing to classic approaches) and the scalability of our solution.

Keywords: Data models, Temporal graphs, Query, Neo4j.

*Corresponding author
Email addresses: landy.andriamampianina@activus-group.fr (Landy Andriamampianina), franck.ravat@irit.fr (Franck Ravat), jiefu.song@activus-group.fr (Jiefu Song), nathalie.valles-parlangeau@ut-capitole.fr (Nathalie Vallès-Parlangeau)

1. Introduction

The era of Big/Smart data has seen a proliferation of highly interconnected data [1]. In response to this, graph data management systems (GDMS) has emerged for managing data in areas where the main concern has to do with the interconnectivity (or topology) of that data [2]. If the temporal dimension of data is taken into account in the context of relational data (such as in multi-version data warehouses [3]), it is not considered in current GDMS. Yet, many kind of changes may occur over time in interconnected data: new interconnections can be added and/or removed and internal characteristics of data can be added and/or removed and/or updated. Moreover, analyzing changes in data can help a decision maker to explain and predict the behaviour of a business domain [4].

At the present time, temporal interconnected data (i.e. interconnected data changing over time) are usually modelled as *temporal graphs* (i.e. graphs with components changing over time) in the area of networks [4]. Such models gather information along time of networks to capture the changes in their structure, such as the formation of new communities in social networks. Few works focus on modelling, querying and storing interconnected and evolving data [2]. Our work focuses on the latter topic. More specifically, the objective of our work is to provide a comprehensive solution for temporal graph data management, ranging from a business oriented (conceptual) data model to an implementation through a systematic study of feasibility and efficiency.

The remainder of this paper is organized as follows. First, we review the literature on temporal data management and graph data management (Section 2). Second, we propose a conceptual model for representing temporal graph data (Section 3). We also provide a graphical notation to facilitate the exploitation of our conceptual model by business users. Third, we propose rules to transform our conceptual model into the logical property-graph model (Section 4). Fourth, we propose a study of the feasibility and usability of our conceptual model

through real business analyses (Section 5). Finally, we evaluate the querying efficiency and scalability of our implemented model through experiments using benchmark and real-world datasets (Section 6).

2. Related Work

To manage interconnected data changing over time, we analyze existing approaches in the field of temporal data management and graph data management.

2.1. Temporal data management

Temporal data management involves all methods and techniques to model, query and store time-varying data (or temporal data) [5]. There is a vast literature on this topic since the 80's focusing on relational database management systems (RDBMS) [6]. In this field, the classic type of data model is the point-based data model [7]. This data model is characterized by two features: (i) snapshots of data are taken in a systematic manner (based on pre-defined rules to capture), and (ii) snapshots are taken in a global manner (capturing all data of the database). As a result, we obtain a representation of the states of data at successive discrete time points. This approach is generally applied in the field of networks to capture the changes of interconnected data over time [8]. It is called the "sequence of snapshots" [9]. However, the snapshot-based approach presents some limitations. First, capturing snapshots systematically do not reflect the changes of data but only their states at specific time points [10]. Second, capturing snapshots globally does not allow for tracking the changes of a single object directly. Moreover, it introduces redundancy of data that do not change over time [11]. Consequently, our objective is to manage temporal interconnected data (i) in more flexible manner to track changes at the evolution rate of each object and (ii) more locally to track all changes at the level of each object and avoid redundancy. To do so, the interval-based data model consists in attaching a time interval to each object which refers to the time when a fact was true in the modelled reality [12]. This approach in the field of networks

consists of attaching a time interval to each graph component (node or edge) [9]. Our proposition extends the last approach.

2.2. Graph data management

Graph data management involves all methods and techniques to model, query and store highly interconnected data [2]. Contrary to other models (such as relational model), graph models focus on representing the interconnectivity (or topology) of data in areas where it is the main concern. Their advantages are (i) to be recognized as one of the most simple, natural and intuitive representation of human knowledge [13] and (ii) to allow for a natural and explicit modelling of data having graph structure [2]. However, few works in this field focus on time-varying graph data (or temporal graph data) contrary to the literature of RDBMS. To do so, we analyze existing works involving aspects of the management of temporal graph data at three levels: conceptual, logical and physical.

Conceptual level. At the conceptual level, most modelling solutions of temporal graph data follow the property graph paradigm [2]. The specific features are: a set of nodes representing entities, a set of edges representing relationships between entities and a set of attributes describing each node or edge. Over time, changes may happen to (i) the topology (i.e. the way nodes and edges are connected), to (ii) the descriptive attribute set within a node or an edge and to (iii) the values of an attribute [8]. Some works focus on one above-mentioned change type [14, 15, 16, 17]. For instance, [14] focuses on capturing the addition or removal of edges only, while [15] captures the addition or removal of both nodes and edges. Some works propose a modelling solution including two change types and their combination [18, 19]. For instance, in addition to the evolution of graph topology, [18] captures the changes in the value of node attributes, while [19] capture the changes in the value of edge attributes. Other works try to take into account all changes types [9, 20]. Their graph model relies on a flat structure by creating new nodes for each descriptive attribute in the nodes of the original

Table 1: Temporal graph models at the conceptual level. *The capital letters in the table are defined as follows: EN= Entities, R= Relationships, P = Point-based data model, I = Interval-based data model, N = Node, E = Edge.*

Model	Time approach	Evolution type			Purpose
		Topology	Attribute value	Attribute set	
Evolving graph [14]	P	E			Mining interesting patterns
Dynamic network [15, 16]	P	N/E			Mining interesting patterns
Dynamic attributed graph [18]	P	E	N		Mining co-evolution patterns
Stream graph [17]	I	N/E			Discovery of graph properties
Attributed Dynamic Graph [19]	I	E	E		Temporal paths discovery
Temporal property graph [9, 20]	I	N/E	N	N	Management of temporal graph data
Our model	I	N/E	N/E	N/E	Management of temporal graph data

graph and ignoring the attributes of edges. So the changes of attribute set and attribute value are not managed for edges. Our objective in this paper is to extend the existing modelling solutions by providing a comprehensive overview of graph data and their changing components. Moreover, our work should cover all possible evolution types of temporal graph data (topology, attribute set and attribute value).

Logical level. At the logical level, property-graph and RDF data models are commonly used in the context of graph data management [2]. Traditionally, the translation between the conceptual level and the logical level is framed by rules such as in the relational databases domain. However, to our knowledge, this translation is implicit in the domain of graphs [14, 16, 15]. No standard is defined at the present time to guarantee a compliant implementation of a conceptual model of temporal graph data at the logical level. Our objective in this paper is to propose standard translation rules of our conceptual model into a logical model.

Physical level. At the physical level, existing works try to maximize the implementation and query efficiency of temporal graph data. We distinguish two research axis in existing works: data redundancy reduction and implementation environment (Table 2). Regarding the first axis, snapshots inevitably introduce data redundancy since consecutive snapshots share in common nodes and

Table 2: Temporal graph models at the physical level.

Research axis	Work	Purpose	Setting
Data redundancy reduction	[21]	Snapshot storage and retrieval, Distribution of historical queries	Centralized
	[22]	Snapshot storage and retrieval	Distributed
	[23]	Snapshot storage and retrieval	Centralized
	[24]	Temporal graph storage and algorithms, Interval-centric model	Distributed
	[25]	Temporal property graph, Interval-centric model, Temporal path queries	Distributed
	[26]	Historical graph storage and analysis, Node-centric model	Distributed
Implementation environment	[27]	Modelling, storing and querying time-varying graphs, Neo4j	Centralized
	[28]	Temporal graph data management system, ACID transactions, Neo4j	Centralized

edges that do not change over time [11]. Processing snapshots causes redundant computation limiting scalability [24]. In response to this issue, [21] proposes a strategy to determine when snapshots should be materialized based on the distribution of historical queries. [22] introduces an in-memory data structure and a hierarchical index structure to retrieve efficiently snapshots of an evolving graph. [23] proposes a framework to construct a small number of representative graphs based on similarity. However, these optimization techniques snapshots always accept some data redundancy. To completely avoid data redundancy, some works recommend to use a data model completely in break with snapshots. However, they are oriented towards distributed computing so do not provide a business-oriented view [24, 25, 26]. Regarding the implementation environment, some works focus on evaluating the performance of graph data stores supporting temporal graphs via experimental assessments. Some experiments rely on RDF triple stores, such as Virtuoso[1] or TDB-Jena[2], to store the evolution of Linked Open Data (LOD) in the Semantic Web area [29, 30]. However, it is already known that graph oriented NoSQL databases are more efficient than RDF triple stores when querying RDF data [31]. It is necessary to see if these NoSQL databases are as efficient in the context of temporal graphs. The authors in [27] use Neo4j to store the time-varying networks and to retrieve specific snapshots. The authors in [28] have developed a graph database man-

[1] `https://virtuoso.openlinksw.com/`

[2] `https://jena.apache.org/documentation/tdb/`

agement system based on Neo4j to support graphs changing in the value of nodes and edges' properties but do not address the change in graph topology. It is necessary to evaluate systematically the performance of graph oriented NoSQL databases by including all change types in the set of benchmark queries.

3. Conceptual modelling

In this section, first we introduce how time is modelled within our conceptual modelling (Section 3.1). Second, we present our temporal graph model (Section 3.2). Third, we describe the evolution management using our proposed concepts (Section 3.3). Finally, we illustrate the proposed concepts in an example (Section 3.4).

3.1. Time

Time can be schematized as a domain denoted by Ω, which is linear and discretized by ordered natural numbers corresponding to their succession in time [24, 25]. Each time point corresponds to an *instant*.

A *time unit* is an atomic increment in time defined by some user [24, 25]. It is defined by a mapping function $T(x) \subset 2^{\mathbb{N}}$. $T(x)$ allows to associate a time interval, indexed by $x \in \mathbb{N}$, to a set of instants (Figure 1). Following the definition in [32], it has the following characteristics:

- $0 \in T(0)$ {each time unit starts from the beginning};
- $\forall i, j \in \mathbb{N}, i \neq j \rightarrow T(i) \cap T(j) = \emptyset$ {two continuous blocks do not overlap};
- $\forall i \in \mathbb{N}, \exists j \in \mathbb{N}$ such that $i \in T(j)$ {each time unit covers the whole timeline, i.e., $\mathbb{N}$}.

The most common units are corresponding to the usual partitions of calendars are: millennium, century, year, month, day, week, hour, second, etc. A time unit can be the partition of another such as days for months.

A *time interval* defines a set of instants between two instant limits in time. We denote it $T = [t_s, t_e[$ where $t_s, t_e \in \Omega$ which indicates a time interval starting

from t_s and extending to but excluding t_e. Therefore, an *instant* is a time interval $T = [t_s, t_e]$ where $t_s = t_e$ and $t_s, t_e \in \Omega$. It has no duration relatively to its time unit. [33] propose *interval relations* that are boolean comparators between time intervals. In our model, we manage *the valid time interval* i.e., the time interval during which a fact is true in the modelled reality [12].

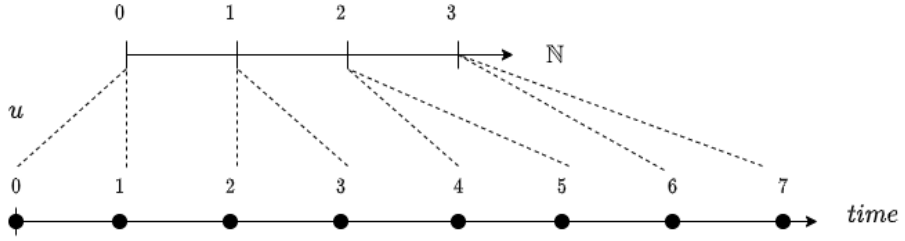


Figure 1: Time modelling.

3.2. Temporal Graph

We define a conceptual model for representing business graph data that change over time [34]. More specifically, we propose concepts to model objects of a business context, the relationships between those objects and their evolution aspects.

Definition 1. *A temporal entity, called* e_i*, is defined by* $\langle l^{e_i}, id^{e_i}, S^{e_i}, T^{e_i} \rangle$ *where* l^{e_i} *is the label of* e_i*,* id^{e_i} *is the identifier of* e_i*,* $S^{e_i} = \{s_1^{e_i}, ..., s_m^{e_i}\}$ *is the non-empty set of states of* e_i *and* T^{e_i} *is the valid time interval of* e_i*. Each state* $s_j^{e_i} \in S^{e_i}$ *is defined by* $\langle A^{s_j}, V^{s_j}, T^{s_j} \rangle$ *where* $A^{s_j} = \{a_1^{e_i}; ...; a_n^{e_i}\}$ *is the set of attributes of* $s_j^{e_i}$*,* $V^{s_j} = \{v(a_1^{e_i}); ...; v(a_n^{e_i})\}$ *is a set of attribute values and* T^{s_j} *is the valid time interval of* $s_j^{e_i}$*. Each* $v(a_q^{e_i}) \in V^{s_j}$ *is the value of each attribute* $a_q^{e_i} \in A^{s_j}$*.*

Definition 2. *The valid time interval of each state of a temporal entity* $s_j^{e_i} \in S^{e_i}$ *is defined by* $T^{s_j} = [t_s, t_e[$ *where* $t_s \neq \emptyset$ *and* $t_e \neq \emptyset$*. The valid time interval of each temporal entity* e_i *is obtained by calculation:*

$$T^{e_i} = \cup_{j=1}^{j=m} T^{s_j} \text{ where } s_j \in S^{e_i} \quad (1)$$

We describe an object of a business context with the concept of *temporal entity* and its descriptive characteristics with the concept of *attributes*. We consider three types of evolution of objects: (i) their presence and absence over time referred as *the evolution in topology*, (ii) the addition and removal of new characteristics referred as *the evolution in attribute set* and (iii) the change in the value of their characteristics referred as *the evolution in attribute value*. So we model a temporal entity through two levels of abstraction: (i) the *topology level* to capture the evolution in its topology and (ii) the *state level* to capture its evolution in terms of attribute set or value.

Definition 3. *A temporal relationship, called* r_i*, is defined by* $\langle l^{r_i}, (s_k, s_j), S^{r_i}, T^{r_i} \rangle$ *where* l^{r_i} *is the label of* r_i*,* (s_k, s_j) *is the couple of entity states* r_i *links,* $S^{r_i} = \{s_1^{r_i}, ..., s_u^{r_i}\}$ *is the non-empty set of states of* r_i *and* T^{r_i} *is the valid time interval of* r_i*. Each state* $s_b^{r_i} \in S^{r_i}$ *is defined by* $\langle A^{s_b}, V^{s_b}, T^{s_b} \rangle$ *where* $A^{s_b} = \{a_1^{r_i}; ...; a_w^{r_i}\}$ *is the set of attributes of* $s_b^{r_i}$ *,* $V^{s_b} = \{v(a_1^{r_i}); ...; v(a_w^{r_i})\}$ *is a set of attribute values and* T^{s_b} *is the valid time interval of* $s_b^{r_i}$*. Each* $v(a_d^{r_i}) \in V^{s_b}$ *is the value of each attribute* $a_d^{r_i} \in A^{s_b}$*.*

Remark 1. *The valid time interval of each state of a temporal relationship* $s_b^{r_i} \in S^{r_i}$ *is defined by* $T^{s_b} \subseteq (T^{s_k} \cap T^{s_j})$ *where* T^{s_k} *is the valid time of the entity state* s_k *and* T^{s_j} *is the valid time of the entity state* s_j*. The valid time interval of each temporal relationship* r_i *is obtained by calculation:*

$$T^{r_i} = \cup_{b=1}^{b=u} T^{s_b} \text{ where } s_b \in S^{r_i} \tag{2}$$

A relationship between two objects of a business context does not have an independent existence. Its existence depends on the objects it links. We describe a relationship between two objects with the concept of *temporal relationship* and its descriptive characteristics with the concept of *attributes*. We consider that relationships can experience the same three types of evolution as objects. Therefore, similar to a temporal entity, we model a temporal relationship through two levels of abstraction.

Definition 4. *L describes a finite set of labels. A label* $l \in L$ *describes the*

semantic of entities (or relationships). So a label groups an entity class (or relationship class). Conversely, an entity (or relationship) has an unique label. Unlabeled entities (or relationships) are semantically indistinct.

As a result of the previous definitions, our *temporal graph* is defined as follows:

Definition 5. *A Temporal Graph, called G, is defined by $\langle E, R, T, \rho, \lambda \rangle$ where:*

- $E = \{e_1, ..., e_g\}$ *is a finite set of temporal entities;*
- $R = \{r_1, ..., r_h\}$ *is a finite set of temporal relationships;*
- T *is the timeline of the temporal graph. It only depends on the valid time intervals of temporal entities as they have an independent existence. So it is obtained by calculation:*

$$T = \cup_{i=1}^{i=g} T^{e_i} \text{ where } e_i \in E \tag{3}$$

- $\rho : R \rightarrow (E \times E)$ *is a function that associates each state of each relationship in R with a pair of entity states in E;*
- $\lambda : (E \cup R) \rightarrow SET^{+}(L)$ *is a function that associates each entity (or relationship) in the temporal graph with a label from L.*

Definition 6. *The temporal graph schema is a tuple $SC = (L^E, L^R, \phi)$ where:*

- $L^E \subset L$ *is a finite set of labels representing the semantic of entities;*
- $L^R \subset L$ *is a finite set of labels representing the semantic of relationships, satisfying that L^E and L^R are disjoint;*
- $\phi : (L^E, L^E) \rightarrow SET^{+}(L^R)$ *is a function that defines the finite and non-empty subset of relationship labels from L^R allowed between a given pair of entity labels.*

We present the graphical notation of our modelling solution in Section 3.4.

3.3. Evolution management

As seen in the conceptual modelling above, we describe the evolution aspects of objects and their relationships of a business context with the concept of *evolution types*: (i) *the evolution in topology*, (ii) *the evolution in attribute set* and (iii) *the evolution in attribute value*. Moreover, in order to keep tracks of these three evolution types, we attach a *valid time interval* at the topology and state levels of a temporal entity (or relationship).

At the topology level, the presence and absence of a temporal entity (i.e., the evolution in topology) is captured by the update of its valid time interval T^{e_i} over time. When an entity e_i is added for the first time in the modelled business context, its valid time interval is $T^{e_i} = [t_{addition}, +\infty)$ where $t_{addition}$ is the time instant of the addition. Otherwise, its valid time interval is updated to $T^{e_i} = \{[t_s, t_e[, ..., [t_{addition}, +\infty)\}$ each time it is added. When an entity e_i is removed from the business context, its valid time interval is updated to $T^{e_i} = \{[t_s, t_e[, ..., [t'_s, t_{removal}[\}$ where $t_{removal}$ is the time instant of the removal and $t'_s < t_{removal}$.

At the state level, a temporal entity evolves according to the addition and/or removal of a new attribute (i.e., the evolution in attribute set) and/or the change in an attribute value (i.e., the evolution in attribute value). A state is related to a set of states that belong to the same entity. Two states of the same entity have different attribute set and/or attribute value. When a new attribute is added/removed or an attribute value changes, a new state of the entity is created instead of overwriting the old state version. The valid time interval of the old state version, s_j, is updated to $T^{s_j} = [t_s, t_{change}[$ where t_{change} is the time instant of the change and $t_s < t_{change}$. The valid time interval of the new state version, s_{j+1}, is $T^{s_{j+1}} = [t_{change}, +\infty)$.

Similar to temporal entities, temporal relationships can evolve in terms of topology, attribute value or attribute set. So to capture the evolution of temporal relationships, we apply the same evolution management.

3.4. Example

For a decision maker, the difficulty of relying on a dataset with temporal interconnected data is to follow how the different information changes over time and how the different information relates to each other. To do so, such dataset can be ideally represented in a temporal graph with our conceptual modelling. In the following, we present the modelling of a dataset of an e-commerce activity into our temporal graph representation and compare its the advantage compared to a snapshot-based representation.

In our business use case, customers view, add to cart and buy items on an e-commerce website. They can make a new action (i.e. view, add to cart and buy) on items each minute. They can modify characteristics of their cart over time by changing items' quantity or by adding a discount code. The website adds new items over time. Moreover, it adds new characteristics to items and updates the value of characteristics over time.

To model a temporal graph representing the e-commerce activity, we propose a two-step approach. The first step consists of identifying the entities and relationships that model the business needs. In our conceptual model, each entity and relationship classes of a business domain are modelled through the concept of labels. The second step consists of identifying the evolution of the various components of the previous schema. In our conceptual model, each entity of a business domain is modelled through the concept of temporal entity. Each relationship of a business domain is modelled through the concept of temporal relationship. All descriptive information of entities and relationships of a business domain are modelled through the concept of attributes. We manage their evolution, in terms of topology - attribute set - attribute value, notably through the concept of states.

In the first step, we identified two entity classes (customer and item) and three relationship classes (view, add to cart and buy). So the formal description of the temporal graph schema is given as follows:

- $L^E = \{CUSTOMER, ITEM\}$

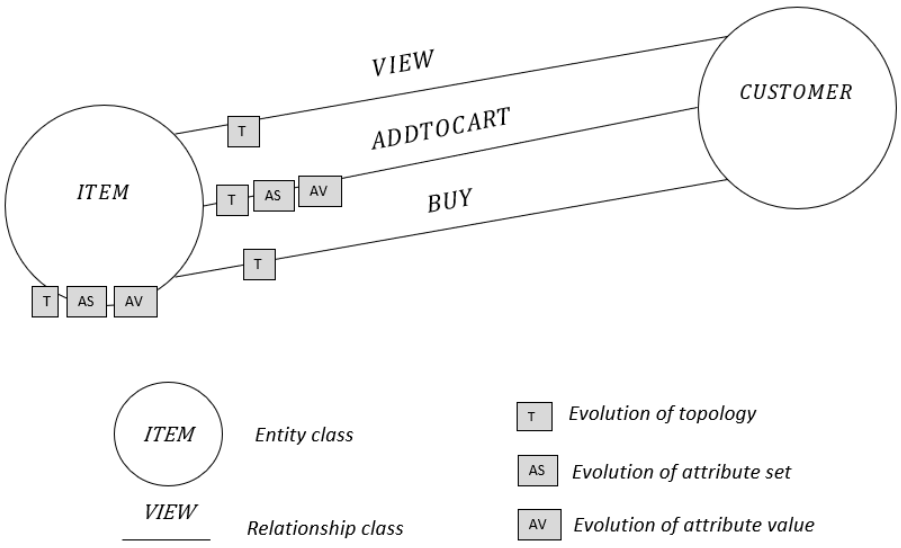


Figure 2: Schema of the e-commerce dataset

- $L^R = \{VIEW, ADDTOCART, BUY\}$
- $\phi(CUSTOMER, ITEM) = \{VIEW, ADDTOCART, BUY\}$

In the second step, we understood that customers do not evolve over time contrary to items. Items evolve over time in terms of their topology, attribute set and attribute value. Similarly, the relationships "add to cart" evolve over time in terms of their topology, attribute set and attribute value. The relationships "view" and "buy" evolve over time in terms of their topology only. We were able to construct the structure of the dataset in Figure 2. According to our conceptual modelling, customers and items become temporal entities. Actions of customers on items (view, add to cart and buy) become temporal relationships. The characteristics of customer, items and carts are translated into attributes. We illustrate in details the modelling of the evolution of these entities and relationships through the following business scenarios in the dataset.

A customer identified $C1$ and called "Smith" never experiences a change in its characteristics since the creation of its account. If we use snapshot-based

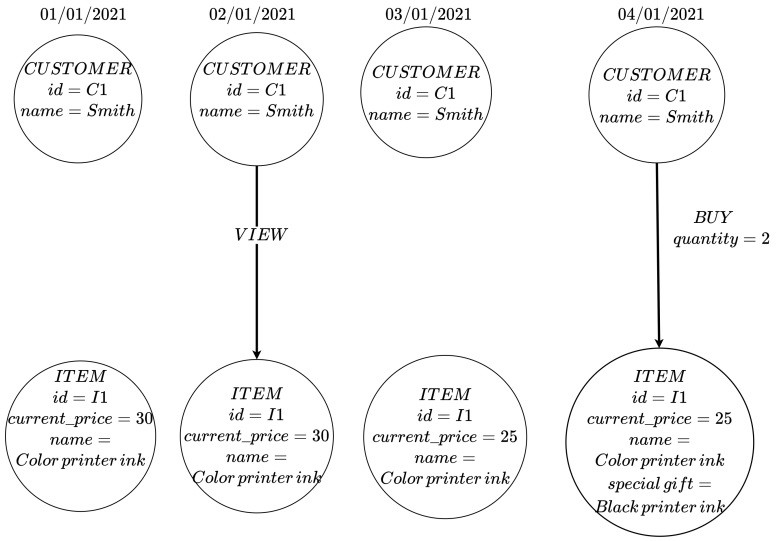


Figure 3: Management of temporal graph data of the example in Section 3.4 with the snapshot-based solution.

approaches, we consider that data are captured at a regular time interval, for instance each day. Therefore, the node representing the customer is repeated at each snapshot as we can see in Figure 3. The advantage of our model is to represent this customer by only one state, numbered 1 in Figure 4, with a start valid date corresponding to the creation date of its account and no ending date. The formal description of this customer according to our conceptual model is given as follows:

$e_1 = \langle CUSTOMER, C1, \{s_1\}, [01/01/2021, +\infty) \rangle$

$s_1 = \langle \{name\}, \{Smith\}, [01/01/2021, +\infty) \rangle$

The website adds new items over time. This concerns notably the item "Color printer ink" identified as $I1$. At its publication on the website, the price of $I1$ is 30. Two days after $I1$'s publication, its *price* has decreased. This refers to the evolution in attribute value of $I1$. One day after, the website has added a new descriptive information ($special\,gift$) to $I1$. This refers to the evolution in attribute set of $I1$. With the snapshot-based approach in Figure 3, as $I1$ does not change during two days, the initial state of $I1$ is repeated in two

snapshots. In our model, a state is generated at each change. The publication of $I1$ generates the first state of $I1$ numbered 2 in Figure 4, with a valid time interval beginning at the date of its publication. The change in $I1$'s attribute value results in the new state, numbered 3, with a valid time interval beginning at the date of the price decrease. The change in $I1$'s attribute set results in the new state numbered 4, with a valid time interval beginning at the date of the attribute addition. Therefore, this produces only 3 nodes for $I1$ in our model instead of 4 nodes in the snapshot-based approach. The formal description of this item according to our conceptual model is given as follows:

$e_2 = \langle ITEM, I1, \{s_2, s_3, s_4\}, \{[01/01/2021, 02/01/2021],$
$[03/01/2021, 03/01/2021], [04/01/2021, +\infty)\}\rangle$

$s_2 = \langle \{current\_price, name\}, \{30, Color\, printer\, ink\}, [01/01/2021, 02/01/2021]\rangle$

$s_3 = \langle \{current\_price, name\}, \{25, Color\, printer\, ink\}, [03/01/2021, 03/01/2021]\rangle$

$s_4 = \langle \{current\_price, name, special\, gift\}, \{25, Color\, printer\, ink, Black\, printer\, ink\}, [04/01/2021, +\infty)\rangle$

Customers can make a new action on items each minute. This refers to the evolution in topology of relationships between customers and items. The customer $C1$ viewed $I1$ once during the day 02/01/2021. During the day 04/01/2021, the customer $C1$ viewed $I1$, added it to cart and then bought it. If we would have adopt the snapshot-based approach, only the last state of data during the day would be kept. As we can see in Figure 3, we lost information about the actions that have been done during the day (view, add to cart). In the contrary, our model keeps all information about the changes during the day. So the two actions $VIEW$ and $ADDTOCART$ are represented by temporal relationships with at least one state in Figure 4.

The characteristics of customer actions can be updated. During the day 04/01/2021, the customer $C1$ has modified the quantity of the item $I1$ in his cart following a *discount code* he received from the website. He has added his *discount code* to his cart. This refers respectively to the evolution in attribute value and set of the relationship $ADDTOCART$ between $C1$ and $I1$. As said previously, these two pieces of information are lost in the snapshot-

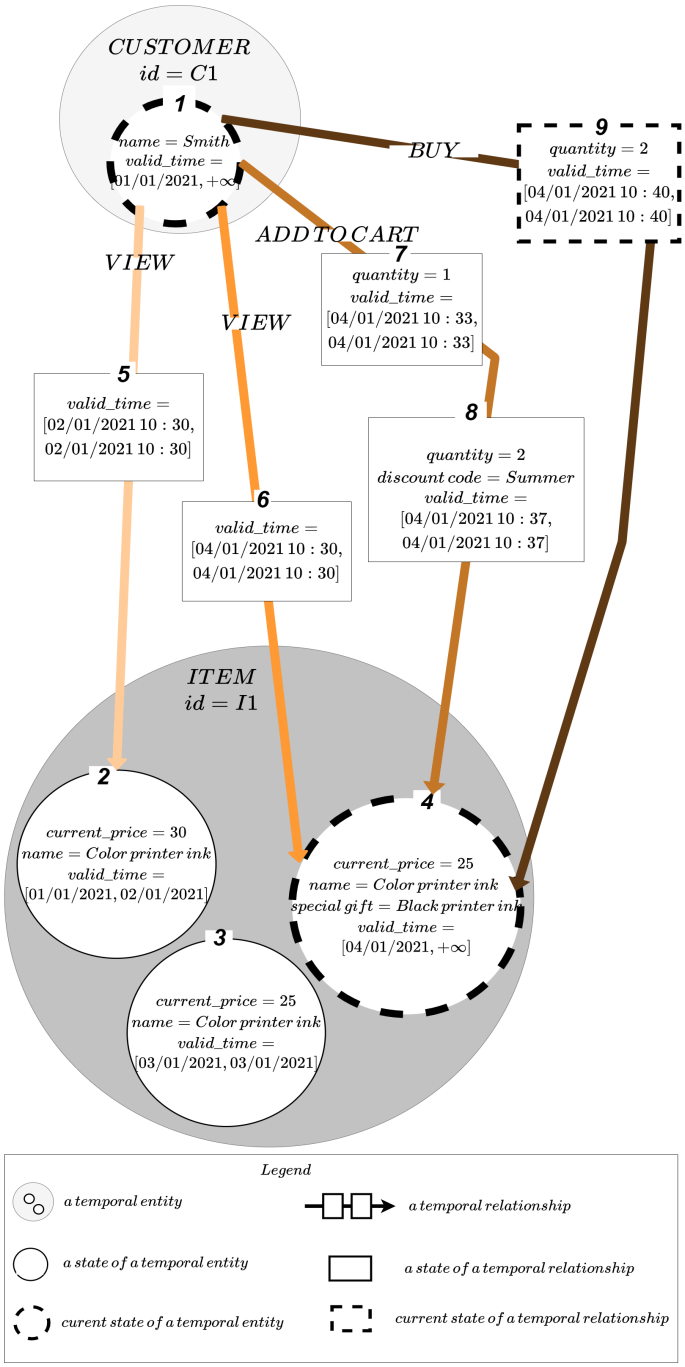


Figure 4: Graphical notation of our temporal graph presented through the example in Section 3.4.

based approach. In our model, the change in the attribute set and value of the relationship $ADDTOCART$ between $C1$ and $I1$ generates two states in Figure 4: (i) a state numbered 7 which is the initial state of the relationship before changes and (ii) a state numbered 8 with one more 1 unit of $I1$'s quantity and a *discount code*. Then, the customer $C1$ bought the item $I1$. This generates the state numbered 9.

The formal description of all customer actions represented is as follows:

$r_1 = \langle VIEW, (s_1, s_2), \{s_5\}, [02/01/2021\,10:30, 02/01/2021\,10:30]\rangle$

$r_2 = \langle VIEW, (s_1, s_4), \{s_6\}, [04/01/2021\,10:30, 04/01/2021\,10:30]\rangle$

$r_3 = \langle ADDTOCART, (s_1, s_4), \{s_7, s_8\}, \{[04/01/2021\,10:33, 04/01/2021\,10:33], [04/01/2021\,10:37, 04/01/2021\,10:37]\}\rangle$

$r_4 = \langle BUY, (s_1, s_4), \{s_9\}, [04/01/2021\,10:40, 04/01/2021\,10:40]\rangle$

$s_5 = \langle \emptyset, \emptyset, [02/01/2021\,10:30, 02/01/2021\,10:30]\rangle$

$s_6 = \langle \emptyset, \emptyset, [04/01/2021\,10:30, 04/01/2021\,10:30]\rangle$

$s_7 = \langle \{quantity\}, \{1\}, [04/01/2021\,10:33, 04/01/2021\,10:33]\rangle$

$s_8 = \langle \{quantity, discount\,code\}, \{2, Summer\}, [04/01/2021\,10:37, 04/01/2021\,10:37]\rangle$

$s_9 = \langle \{quantity\}, \{2\}, [04/01/2021\,10:40, 04/01/2021\,10:40]\rangle$

Through this example, the following advantages of our conceptual model are retained. First, our conceptual model provides a comprehensive overview on the temporal evolution of a dataset for a decision maker. Indeed, it captures in a finer way the evolution of entities and relationships at different levels: topology, attribute set and attribute value. Moreover, it provides a graphical notation that allows for easily representing the topology of data (interconnections of data), the data embedded within the topology and their temporal evolution. Second, our conceptual model represents the temporal evolution of a dataset in a synthetic manner. No information is lost or redundant after the modelling process. If we would have adopt the snapshot-based approach, we would have 4 more nodes than our model. Finally, our conceptual model is flexible in the way of modelling business requirements. For instance, suppose now that the dataset includes a customer that adds to cart the same item (same state of item) at

two different dates for two different orders. We can easily do that by adding an attribute "order" for each state of $ADDTOCART$ relationship between the same customer (same state of customer) and the same item (same state of item). We illustrate this case in Figure 5.

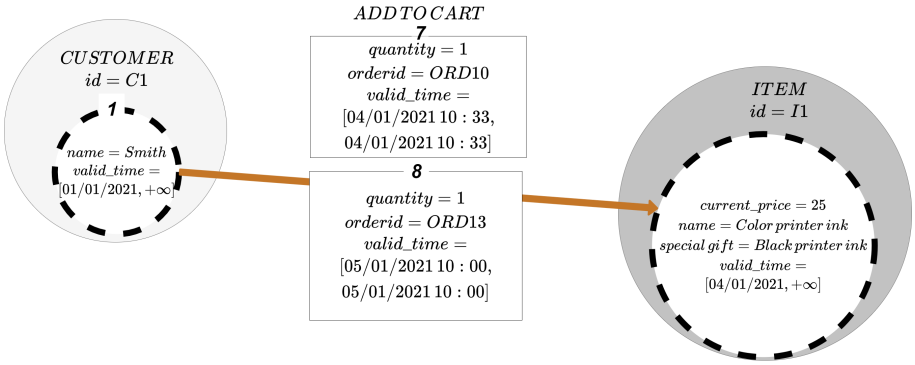


Figure 5: Extended example.

4. Logical modelling

The objective of logical modelling is to take into account the type of data storage chosen for the implementation. In our case, we chose the logical property graph model because most of graph-oriented NoSQL data stores, such as Neo4j, are designed to store property graphs. Our objective is then to translate our conceptual representation of temporal graph into a logical property graph.

According to [35], a property graph is defined as $PG = \langle N, D, \eta, \Lambda, \Sigma \rangle$ where N is a finite set of nodes (also called vertices), D is a finite set of edges, $\eta : D \rightarrow (N \times N)$ is a function that associates each edge in D with a pair of nodes in N, $\Lambda : (N \cup D) \rightarrow SET^{+}(L)$ is a function that associates a node/an edge with a set of labels from L, and $\Sigma : (N \cup D) \times P \rightarrow SET^{+}(V)$ is a function that associates nodes/edges with properties. Each property is a key-value pair (p, v) where p is the property name and v the property value.

We propose a transformation process between our conceptual temporal graph

Temporal graph	Property graph
a state of a temporal entity s_j	a node
a state of a temporal relationship s_b	an edge
a valid time interval of an entity state T^{s_j}	two properties*
a valid time interval of a relationship state T^{s_b}	two properties*
a temporal entity e_i	a set of nodes (with different valid time intervals)
a temporal relationship r_i	a set of edges (with different valid time intervals)
a valid time interval of a temporal entity T^{e_i}	by query
a valid time interval of a temporal relationship T^{r_i}	by query
a label of a temporal entity l^{e_i}	a label
a label of a temporal relationship l^{r_i}	a label
a temporal entity's identifier id^{e_i}	a property
an attribute of a temporal entity $a_q^{e_i}$	a property
an attribute of a temporal relationship $a_d^{r_i}$	a property

Table 3: Transformation rules of our conceptual model into the logical model of property graph. *$startvalidtime$ and $endvalidtime$.

and a logical property graph via a generic algorithm (Algorithm 1). The transformation process receives our temporal graph G as input and returns the property graph PG. For each state s of each temporal entity e in G, 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. For each state s of each temporal relationship r in G, 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. As a result of the Algorithm 1, we obtain the transformation rules presented in Table 3.

We graphically illustrate this transformation process through the mapping of the temporal graph in Figure 4 into the property graph in Figure 6. The resulting property graph is composed of 4 nodes and 5 edges. We notice that for the item in Figure 4, 3 nodes are needed to represent its changes. Similarly, we observe that 2 edges are required to represent the changes of the $ADDTOCART$ relationship.

Algorithm 1: Mapping algorithm: from conceptual temporal graph to logical property graph

Input: Temporal Graph: $G = \langle E, R, T, \rho, \lambda \rangle$

Output: Property Graph $PG = \langle N, D, \eta, \Lambda, \Sigma \rangle$

`/* create the Property Graph */`

$N \leftarrow \emptyset$

$D \leftarrow \emptyset$

foreach *temporal entity* $e_i \in E$ **do**

  $nodeLabel \leftarrow getEntityLabel(e_i)$

  $p_{id} \leftarrow createProperty(\text{“}id\text{”}, getEntityId(e_i))$

  foreach *state* $s_j \in S^{e_i}$ **do**

    $nodeProperties \leftarrow \emptyset$

    $p_{Tstart} \leftarrow$
    $createProperty(\text{“}startvalidtime\text{”}, getStartValue(T^{s_j}))$

    $p_{Tend} \leftarrow createProperty(\text{“}endvalidtime\text{”}, getEndValue(T^{s_j}))$

    $nodeProperties \leftarrow nodeProperties \cup \{p_{id}, p_{Tstart}, p_{Tend}\}$

    foreach *attribute* $a \in A^{s_j}$ **do**

      $p_{att} \leftarrow createProperty(\text{“}a\text{”}, getAttributeValue(a))$

      $nodeProperties \leftarrow nodeProperties \cup \{p_{att}\}$

    `/* create a node with a set properties */`

    $N \leftarrow N \cup createNode(nodeLabel, nodeProperties)$

foreach *temporal relationship* $r_i \in R$ **do**

  $edgeLabel \leftarrow getRelationshipLabel(r_i)$

  $nodeStartLabel \leftarrow getEntityLabel(s_k)$

  $nodeEndLabel \leftarrow getEntityLabel(s_j)$

  $nodeStartProperties \leftarrow$
  $\{(\text{“}id\text{”}, getEntityId(s_k)), (\text{“}startvalidtime\text{”}, getStartValue(s_k)),$
  $(\text{“}endvalidtime\text{”}, getEndValue(s_k))\} \cup$
  $getAllAttributeValuePair(s_k)$

  `/* getAllAttributeValuePair() returns the set of all attribute-value pairs of a state */`

```
24
25    nodeEndProperties ←
        {("id", getEntityId(s_j)), ("startvalidtime", getStartValue(s_j)),
26    ("endvalidtime", getEndValue(s_j))} ∪
        getAllAttributeValuePair(s_j)
27    nodeStart ←
        matchNode(PG, nodeStartLabel, nodeStartProperties)
28    nodeEnd ← matchNode(PG, nodeEndLabel, nodeEndProperties)
29    foreach state s_b^{r_i} ∈ S^{r_i} do
30        edgeProperties ← ∅
31        p_{Tstart} ←
            createProperty("startvalidtime", getStartValue(T^{s_j}))
32        p_{Tend} ← createProperty("endvalidtime", getEndValue(T^{s_j}))
33        edgeProperties ← edgeProperties ∪ {p_{Tstart}, p_{Tend}}
34        foreach attribute pair a ∈ A^{s_b} do
35            p_att ← createProperty("a", getAttributeValue(a))
36            edgeProperties ← edgeProperties ∪ {p_att}
          /* create an edge with a set properties                    */
37        D ← D ∪
            createEdge(nodeStart, nodeEnd, edgeLabel, edgeProperties)
```

5. Analysis of evolving data using temporal graph: a use case

After discussing the conceptual representation of temporal interconnected data (Section 3) and the way to implement it (Section 4), we now present a case
410 study of its implementation in a graph-oriented NoSQL data store based on the dataset of Figure 4 to demonstrate its technical feasibility and usability.

To evaluate the technical feasibility of our conceptual model, we applied the mapping process in Algorithm 1 to implement the dataset in Figure 4 in Neo4j

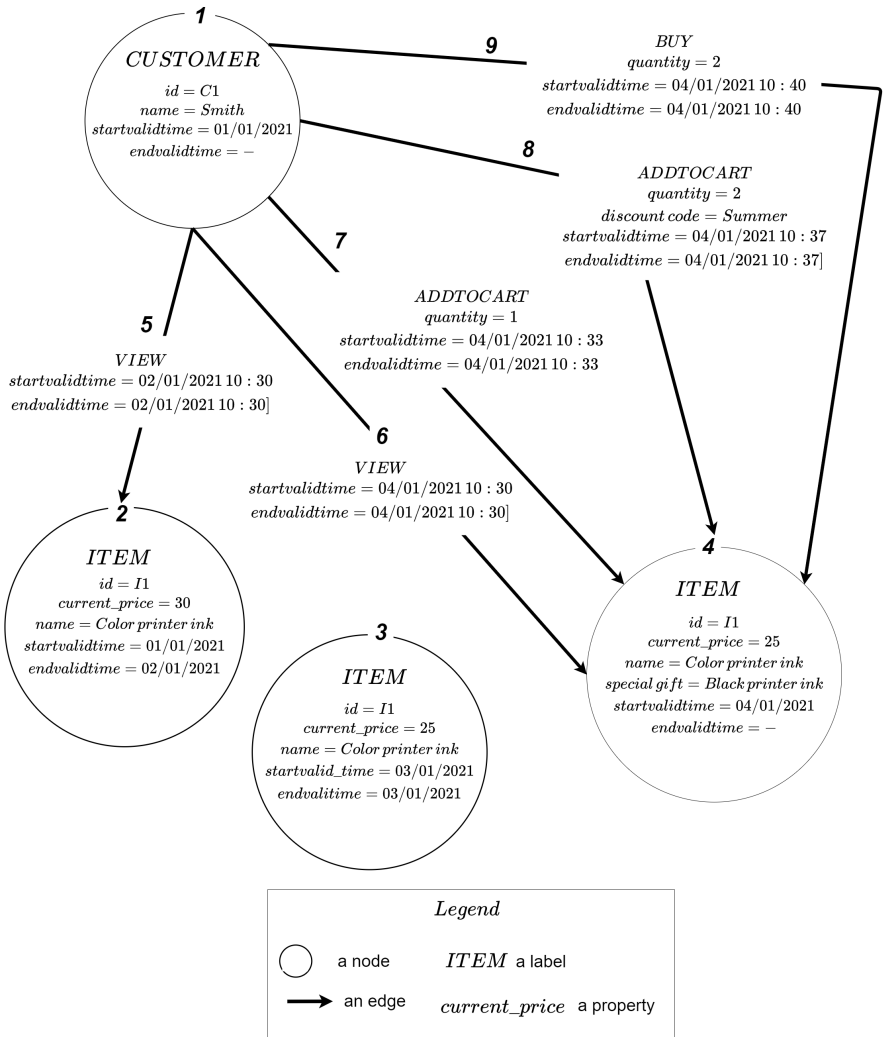


Figure 6: Translation of the conceptual temporal graph in Figure 4 into the logical property graph.

[3], a graph data store supporting the property graph model. Figure 7 presents the result of this implementation.

To evaluate the usability of our conceptual model, we identified the possi-

[3] `https://neo4j.com/`

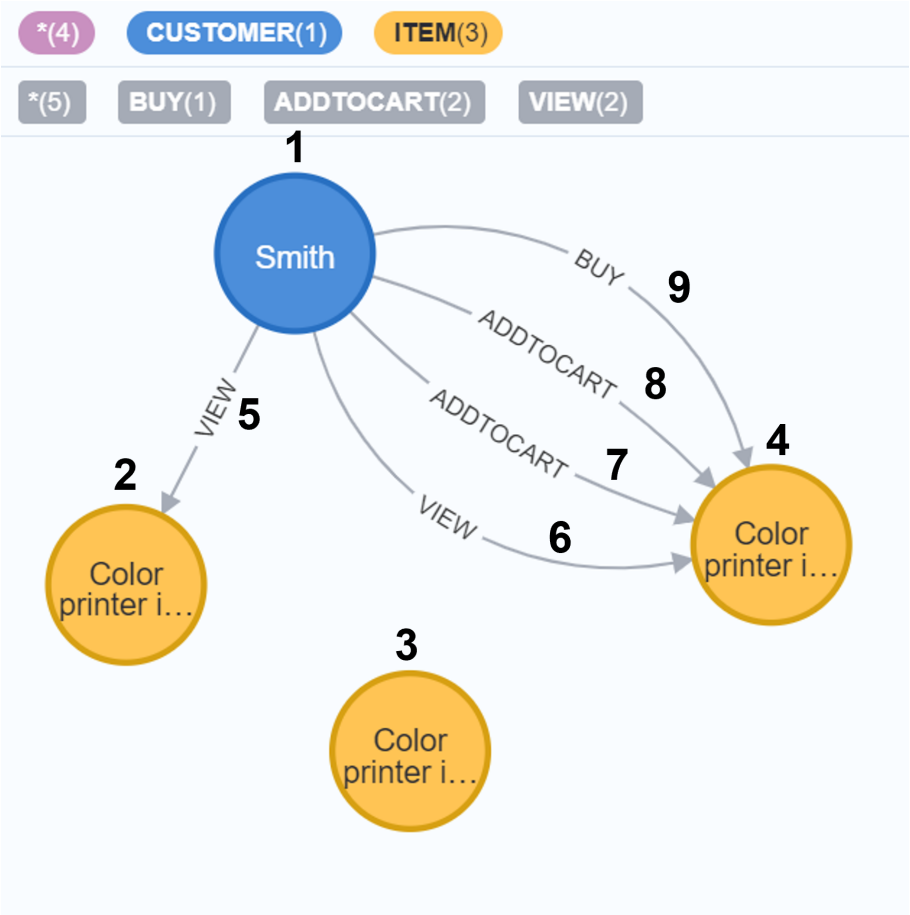


Figure 7: Implementation of the dataset in Figure 4 in Neo4j.

ble analyses on our temporal graph representation. Capturing temporal data evolution provides complementary viewpoints (or perspectives) to explain the "why" underlying observed phenomena and behaviors. First, the user can make a classic analysis according to the graph component only (e.g. entities and/or relationships) without temporal dimension. Second, the user can make an analysis according to time dimension (e.g. on continous or non continous periods etc.). Third, the user can make an analysis according to the evolution type (attribute set, attribute value or topology). The user can cross these different

analysis lines to obtain more valuable insights. In the following, we propose several cross-analyses of the e-commerce dataset.

Decision makers make a first business analysis (B1). The 4th january at 10:00, the e-commerce company announces a discount code on the website homepage for a summer promotion. To evaluate the impact of this announcement, they want to know if customers use the discount code in the hour following the announcement. This consists in analyzing the addition of the attribute *discount_code* in the attribute set of the states of $ADDTOCART$ relationships during the hour. This is translated in Cypher, the language query of Neo4j as follows:

```
MATCH (c:CUSTOMER)-[r:ADDTOCART]->(i:ITEM)
WHERE datetime(r.startvalidtime)< datetime("2021-01-04T12:00")
AND datetime(r.startvalidtime)>=datetime("2021-01-04T10:00")
RETURN  c.id + "-"+ i.id as relationshipCUSTOMERITEM,
collect({time:datetime(r.startvalidtime),
attributeset:keys(r)}) as statesofADDTOCART
```

As a result of B1, we obtain in Figure 8 the attribute set of all states of $ADDTOCART$ relationships in the dataset for the period of interest. We observe that the customer identified $C1$ has used the discount code to buy the item identified $I1$ at 10:37 after the announcement of the summer promotion.

Decision makers make a second business analysis (B2). To evaluate the impact of the previous announcement, they also want to know if the quantity of items added by customers in their card has changed in the hour following the announcement. This consists in analyzing the changes through time of the value of the attribute *quantity* of the states of $ADDTOCART$ relationships during the hour. This is translated in Cypher as follows:

Figure 8: Result of business analysis B1.

```
MATCH (c:CUSTOMER)-[r:ADDTOCART]->(i:ITEM)
WHERE datetime(r.startvalidtime)< datetime("2021-01-04T12:00")
AND datetime(r.startvalidtime)>=datetime("2021-01-04T10:00")
RETURN  c.id + "-"+ i.id as relationshipCUSTOMERITEM,
collect({time:datetime(r.startvalidtime), quantity:r.quantity})
as statesofADDTOCART
```

As a result of B2, we obtain in Figure 9 the value of the attribute *quantity* for each $ADDTOCART$ relationship in the dataset for the period of interest. We observe that the customer $C1$ has updated the quantity of item $I1$ in his

Figure 9: Result of business analysis B2.

cart at 10:37 after the announcement of the summer promotion.

Decision makers make a third business analysis (B3). The e-commerce website records the highest number of sales on item $I1$. They want to know if this increase in $I1$'s sales is due to changes in its characteristics (new price, offer or picture etc.). This consists in analyzing the changes that occurred between the states of the temporal entity $I1$. This is translated in Cypher as follows:

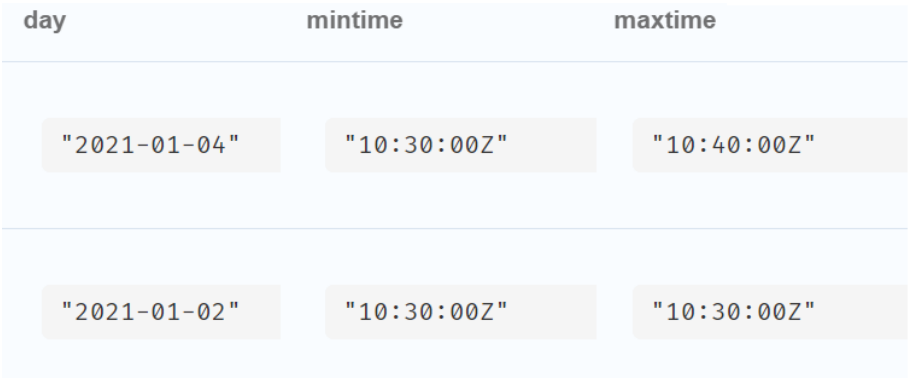
```
MATCH (i:ITEM)
WHERE i.id="I1"
WITH collect(i) as lists
UNWIND range(0,(size(lists)-2)) as j
RETURN "state at " + lists[j].startvalidtime+" AND "
+ "state at " +lists[j+1].startvalidtime as states,
apoc.diff.nodes(lists[j], lists[j+1]) as changesbetweenstates
```

As a result of B3, we obtain in Figure 10 the changes that occur between

Figure 10: Result of business analysis B3.

$I1$'s states in terms of attribute set and value. For instance, we observe that the price of $I1$ has changed from the 1st january to the 3rd january. Then, from the 3rd january to the 4th january, the attribute $special\_gift$ has been added to $I1$.

Decision makers make a fourth business analysis (B4). They want to know the time period in which customers are active on the website each day. It



day	mintime	maxtime
"2021-01-04"	"10:30:00Z"	"10:40:00Z"
"2021-01-02"	"10:30:00Z"	"10:30:00Z"

Figure 11: Result of business analysis B4.

contributes to customer profiling to make more adapted policies in the future. This consists in analyzing the addition and removal of temporal entities and relationships to see when they are connected and disconnected from the website. This is translated in Cypher as follows:

```
MATCH (c:CUSTOMER)-[r]->(i:ITEM)
RETURN date(r.startvalidtime) as day,
min(time(r.startvalidtime)) as mintime,
max(time(r.endvalidtime)) as maxtime
```

As a result of B4, we obtain in Figure 11 for each day, the minimum start valid time and the maximum end valid time at which customers connect to the website.

To sum up, our solution facilitates the exploration of temporal graph data through our conceptual model. Indeed, we identify directly in the implementation of our temporal graph all change data (Figure 7). Moreover, our solution provides a straightforward data restitution without introducing modelling-related concepts in the analysis results visualization (Figures 8, 9 and 10, 11). In this way, this solution makes the technical complexity transparent to non-expert users.

6. Experimental evaluation

6.1. Protocol

6.1.1. Objectives

We run two series of experiments with the two following objectives:

- to evaluate the efficiency of our proposed temporal graph model by comparing its storage and query performance to the snapshot-based temporal graph.
- to evaluate the scalability of our proposed model by comparing its query performance on different data volumes (Section 6.3).

6.1.2. Methods

To avoid any bias in datasets, we need to include both benchmark and real datasets. These datasets should provide different scale factors in order to measure scalability. Moreover, we need benchmark queries with a complete coverage of different analysis types (see Section 6.1.4): temporal analysis (according to different time granularity), graph scope (from subgraph to entire graph) and evolution analysis (according to attribute value, attribute set or topology).

For our two specific objectives (Section 6.1.1), we conducted two series of experiments according to the following methods. The first series consists of a comparative study of their storage and query performance through the size, creation time and query execution times (see Section 6.2). We implemented two snapshot-based approaches to compare our model performance. The second series consists of evaluating the scalability of our model during querying real-world datasets (see Section 6.3).

6.1.3. Datasets

TPC-DS datasets. Temporal evolutions exist in a reference benchmark available online, namely TPC-DS benchmark[4]. This benchmark is based on transaction

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

data of a retail company. It allows us to find all the three types of evolution: (i) attribute value, (ii) attribute set and (iii) topology. We used the dataset from this benchmark to answer the objective of evaluating the efficiency of our model compared to the snapshot-based model. To do so, we transformed the generated dataset from the benchmark into three datasets having our temporal graph, a classic snapshots and an optimized snapshots representations. All transformation details of the TPC-DS dataset into the three representations are available on the website `https://gitlab.com/2573869/dke_temporal_graph_experiments`. In Table 4, we present the results of the transformation steps: the number of nodes/edges/snapshots and the evolution types of the three TPC-DS datasets.

E-commerce dataset. The E-commerce dataset has been collected from a real-world ecommerce website by RetailRocket company. It is available on Kaggle [5]. It is about customers' activity on the website (views, add to cart and transactions). The dataset includes changes over time: (i) on item characteristics with the addition of new attributes over time and the change in attribute value but also (ii) on the interactions between customers and items like clicks, add to carts and transactions. We used this dataset to answer the objective of evaluating the scalability of our model. To do so, we transformed the E-commerce dataset into our temporal graph representation. All transformation details are available on the website `https://gitlab.com/2573869/dke_temporal_graph_experiments`. In Table 4, we present the result of the transformation steps: the number of nodes/edges and the evolution types of E-commerce dataset.

Social experiment dataset. The Social experiment dataset has been collected from a social experiment on students from MIT who lived in dormitory [36]. It is available online at Reality Commons website[6]. This dataset includes changes

[5] `https://www.kaggle.com/retailrocket/ecommerce-dataset?select=item_properties_part2.csv`)

[6] `http://realitycommons.media.mit.edu/socialevolution.html`

over time: (i) on the value of the symptoms of students and (ii) on the interactions between students. We used this dataset to answer the objective of evaluating the scalability of our model. To do so, we transformed the Social experiment dataset into our temporal graph representation. All transformation details are available on the website `https://gitlab.com/2573869/dke_temporal_graph_experiments`. In Table 4, we present the result of the transformation steps: the number of nodes/edges and the evolution types of Social experiment dataset.

Citibike dataset. The Citibike dataset is provided by the company Citibike. The Citibike company collects data about their bicycle rentals since the year 2013 in New York City and makes them avalaible online[7]. This dataset includes bike stations and trips between these stations. We identified that the attribute set describing trips between stations has changed since May 2021. Moreover, the value of the attributes describing trips changes over time. We used this dataset to answer the objective of evaluating the scalability of our model. To do so, we transformed the Citibike dataset into our temporal graph representation. All transformation details are available on the website `https://gitlab.com/2573869/dke_temporal_graph_experiments`. In Table 4, we present the result of the transformation steps: the number of nodes/edges and the evolution types of Citibike dataset.

6.1.4. Benchmark queries

To conduct our two series of experiments, we used the same benchmark queries to evaluate the query performance. To do so, we identified the possible query types according all analysis axes and sub-axes that a decision-maker could have when querying temporal graph data [26, 37]. The first analysis axis is the graph component to evaluate the cost of querying data at the level of a single entity (SE) or a set of related entities (SU) or the entire graph (G). The second analysis axis is the evolution type to evaluate the cost of querying changes in

[7] `https://www.citibikenyc.com/system-data`

Implementation	TPC-DS: Temporal graph	TPC-DS: Classic snapshots	TPC-DS: Optimized snapshots	E-commerce	Social experiment	Citibike
Objective of efficiency evaluation	Y	Y	Y	N	N	N
Objective of scalability evaluation	N	N	N	Y	Y	Y
# Nodes	112 897	7 405 461	5 347 477	4 821 694	33 934	2 861
# Edges	1 693 623	4 207 657	4 044 481	5 222 996	2 168 270	27 561 618
# Snapshots	N/A	60	53	N/A	N/A	N/A
Evolution types of entities	AV, AS, T	AV, AS, T	AV, AS, T	AV, AS, T	AV, T	∅
Evolution types of relationships	AV, AS, T	AV, AS, T	AV, AS, T	T	T	AV, AS, T

Table 4: Characteristics of datasets. *Y= Yes, N= No, AV = Attribute Value, AS = Attribute Set, T = Topology.*

data in terms of: attribute set (AS), attribute value (AS) or topology (T). The third analysis axis is the time scope to evaluate the cost of querying data at the level of a single time point (SP), a single interval (SI), multiple time points (MP) or multiple time intervals (MI). The fourth analysis axis is the operation type used: (i) comparison aiming at evaluating how does a graph component change over time with respect to a temporal evolution type (C) and (ii) aggregation aiming at evaluating an aggregate function (A).

Then, we created benchmark queries in Table 5 by crossing the different sub-axes of analysis to distribute possible query scenarios in a balanced way. Each benchmark query represents a possible combination of analysis sub-axes. As a result, we obtained 28 queries. Finally, we translated these benchmark queries in the native query language of Neo4j: Cypher.

6.1.5. Technical environment

We used the same hardware configuration for the two experiments. It is as follows: PowerEdge R630, 16 CPUs x Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40Ghz, 63.91 GB. One virtual machine is installed on this hardware. This virtual machine has 6GB in terms of RAM and 100GB in terms of disk size. We installed on this virtual machine Neo4j (community version 4.1.3) as data store for our datasets. To avoid any bias in the disk management and query performance, we did not use any customized optimization techniques but relied

		Graph component	Evolution type	Time scope	Operation type
Q1	The descriptive attributes of an entity at X	SE	AS	SP	
Q2	The descriptive attributes of an entity at X and Y	SE	AS	MP	
Q3	The changes that occurred on the descriptive attributes of an entity between X and Y	SE	AS	MP	C
Q4	The descriptive attributes of an entity from X to Y	SE	AS	SI	
Q5	The descriptive attributes of an entity at a regular period	SE	AS	MI	
Q6	The changes that occurred on descriptive attributes of an entity from X to Y	SE	AS	SI	C
Q7	The value of of an entity attribute at X	SE	AV	SP	
Q8	The value of an entity attribute at X and Y	SE	AV	MP	
Q9	The change in the value of an entity attribute between X and Y	SE	AV	MP	C
Q10	The value of an entity attribute from X to Y	SE	AV	SI	
Q11	Aggregation on the value of an entity attribute at a regular period	SE	AV	MI	A
Q12	A subgraph at X	SU	T	SP	
Q13	A subgraph at X and Y	SU	T	MP	
Q14	Aggregation on a subgraph at X	SU	T	SP	A
Q15	Aggregation on a subgraph at X and Y	SU	T	MP	A
Q16	A subgraph from X to Y	SU	T	SI	
Q17	A subgraph at a regular period	SU	T	MI	
Q18	Aggregation on a subgraph at a regular period	SU	T	MI	A
Q19	The descriptive attributes of a relationship at X	SU	AS	SP	
Q20	The descriptive attributes of a relationship at X and Y	SU	AS	MP	
Q21	The changes that occurred on the descriptive characteristics of a relationship between X and Y	SU	AS	MP	C
Q22	The descriptive attributes of a relationship from X to Y	SU	AS	SI	
Q23	The changes that occurred on the descriptive characteristics of a relationship from X to Y	SU	AS	SI	C
Q24	The value of a relationship attribute at X	SU	AV	SP	
Q25	The value of a relationship attribute at X and Y	SU	AV	MP	
Q26	The value of a relationship attribute from X to Y	SU	AV	SI	
Q27	Aggregation on the the value of a relationship attribute at a regular period	SU	AV	MI	A
Q28	The state of the entire graph at X	G		SP	

Table 5: Benchmark queries. X and Y describe time points defined on a time unit. *SE = Single Entity, SU = Subgraph, G = Entire Graph, AS = Attribute Set, AV = Attribute Value, T = Topology, SP = Single Point, MP = Multiple Points, SI = Single Interval, MI = Multiple Intervals, C = Comparison, A = Aggregation.*

on default tuning of Neo4j.

6.1.6. Summary

Regarding the first series of experiments, we created, for each of the three benchmark datasets (TPC-DS), 28 queries according to the query types we set in Section 6.1.4. We run each query ten times and took the mean time of all runs as final execution time. This makes a total of 84 queries ($28\,queries \times 3\,datasets$) and 840 executions ($28\,queries \times 3\ datasets \times 10\,times$).

Regarding the second series of experiments, we have three scale factors from 0.3GB to 6.7GB. We created a total of 44 queries adapted to the business contexts of the three real datasets (E-commerce, Social experiment and Citibike). We run each query ten times and took the mean time of all runs as final execution time. This makes a total of 440 executions ($44\,queries \times 10\,times$).

6.2. Results of the efficiency evaluation of our model

For this first series of experiments, we used the three TPC-DS datasets having respectively our temporal graph, classic and optimized snapshots representations (Section 6.1.3). The classic snapshots consists in sampling graph data at a regular time period (here we chose a month). Our optimized snapshots approach consists in creating a snapshot only if it includes a change compared to a previous snapshot. We compared the storage and query performance of our temporal graph implementation to classic and optimized snapshot-based implementations through the size, creation time and query execution times in Neo4j. The query execution time is the elapsed time in seconds for processing the query. We run the 28 benchmark queries for each implementation (Section 6.1.6).

Observations of storage performance. In Table 6, we observe that our model reduces respectively by 12 times and 9 times the size of database instance storing classic snapshots and optimized snapshots. Moreover, the datasets based on snapshot approaches require more time to be imported since they contain more nodes and edges than our model.

Implementation	TPC-DS : Temporal graph	TPC-DS: classic snapshots	TPC-DS: Optimized snapshots
Size (in GB)	0,3	3,7	2,8
Creation time (in sec)	15,795	56,529	45,827

Table 6: Size and creation time of graph database instances in Neo4j based on benchmark datasets.

Observations of query performance. In Figure 12, we observe the execution times for processing each benchmark query in Table 5. Queries Q1-Q6 are instantaneous (close to 0) for the three implementations. Q17-Q21 and Q27 record execution spikes for the classic and optimized snapshots implementations. The execution time of Q28 explodes for the classic snapshots and temporal graph implementations. Q28 runs out of memory for the optimized snapshots implementation. The rest of benchmark queries (Q7-Q16 and Q22-Q26) does not exceed 6 seconds for the three approaches. Overall, the execution query times of the temporal graph are lower than both snapshot-based approaches.

In Figure 13, we observe the average gain in execution times of the temporal graph implementation over both snapshots implementations by query types. First, we analyze the query performance of our temporal graph according to the graph component, that is requesting information at the level of a single entity (SE), a subgraph (SU) or the entire graph (G). We observe that the temporal graph implementation outperforms both snapshots approaches by saving 92%-93% of their average execution times on querying a single entity or subgraph. The gain of the temporal graph over the classic snapshots on querying the entire graph is smaller accounting for 35%.

Second, we analyze the query performance of our temporal graph according to the evolution type, that is requesting information at the level of attribute set (AS), attribute value (AV) or topology (T). We observe that the gain of the temporal graph implementation is the highest (99%) on querying attribute

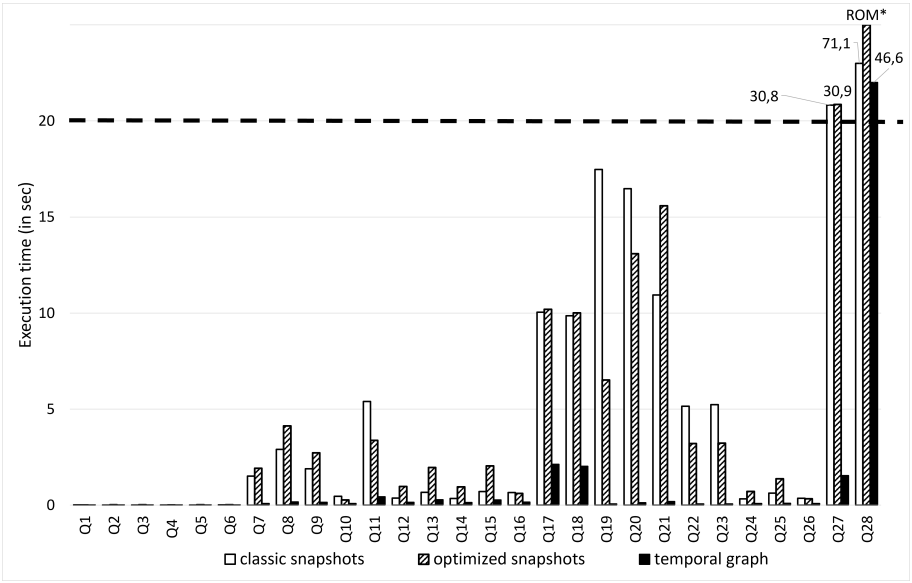


Figure 12: Execution times of 28 benchmark queries. *ROM = Run Out of Memory.

set over both snapshot-based implementations. Regarding queries on attribute value, the temporal graph allows us to save 94% of average execution times over both snapshot-based implementations. The execution times gain of the temporal graph over both snapshots implementation is smaller on querying topology: 77% over classic snapshots and 81% over optimized snapshots.

Third, we analyze the query performance of our temporal graph according to the time scope, that is requesting information at the level of a single time point (SP), multiple time points (MP), a single interval (SI), or multiple time intervals (MI). We observe that there is no big difference of execution times gain of the temporal graph implementation between querying a single time point, multiple time points or single interval. The temporal graph saves from 94% to 97% of the average query execution times of snapshot-based implementations. However, the temporal graph saves less execution times on querying multiple intervals (89%).

Last but not least, we focus on the query performance of our temporal graph according to the operation type, that is comparison (C) or aggregation (A). We

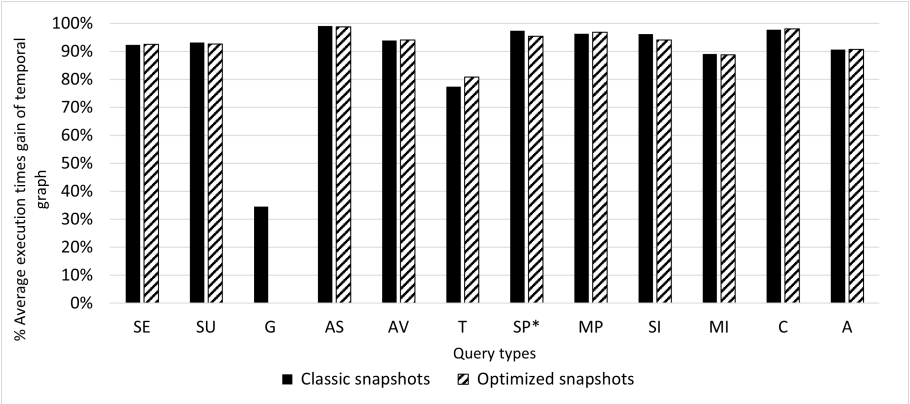


Figure 13: Average execution times gain (in %) of our temporal graph over classic and optimized snapshots by query types. *We do not take into account the execution time of Q28 in the computation of average execution time of SP queries because it explodes or runs out of memory for each implementation. *SE = Single Entity, SU = Subgraph, G = Entire Graph, AS = Attribute Set, AV = Attribute Value, T = Topology, SP = Single Point, MP = Multiple Points, SI = Single Interval, MI = Multiple Intervals, C = Comparison, A = Aggregation.* .

observe that the temporal graph saves more execution times of both snapshots implementations for processing comparison (98%) than aggregation operations (91%).

Discussion. The gap in the query performance between the temporal graph and the two snapshots based implementations is partly due to difference of the data volume involved in queries. The two snapshots approaches use a different time management method than our model. This leads to larger use of disk space (Table 6) and more time to process during querying (Figure 12). Across all query types, the temporal graph implementation always outperforms both snapshot-based implementations (Figure 13). Though the optimized snapshots implementation consumes less disk space than classic snapshots implementation, our temporal graph saves almost the same average query execution times over both snapshot-based implementations. Indeed, as time is managed differently in the two snapshot-based models, it is also queried differently. Conditions on time for classic snapshots are translated in Cypher by simple time predicates.

Conversely, conditions on time in queries for the optimized snapshots are translated in Cypher by a sub-query to search for the snapshot that is the closest a requested time. So this is why the query performance of the optimized snapshots implementation reaches execution times almost equal or higher (e.g. Q13, Q14 or Q21) than the classic snapshots implementation.

Implications. The choice of a data model to manage evolving data impacts significantly the storage and querying efficiency. Our model has a double advantage. First, it allows to get rid of data redundancy. So it saves a significant amount of space on the disk compared to snapshots. Second, it supports efficiently a wide range of queries while keeping average execution times low. The implementation with our model allows to save up to 99% of execution times compared to both snapshot-based implementations.

6.3. Results of the scalability evaluation of our model

For this second series of experiments, we used three real datasets (Social Experiment, E-commerce and Citibike) representing three different scales of data volume and having our temporal graph representation (Section 6.1.3). We compared the query performance of the three implementation according to their scale factors: the size, number of nodes and edges. More precisely, we analyzed (i) the execution times of queries involving only entities (SE) at three different scales of the number of nodes and (ii) the execution times of queries involving relationships (SU) at three different scales of the number of edges. These two analyses allow us to get an idea of the impact of the growing size and interconnectivity of a dataset. We were not able to run the 28 benchmark queries for each implementation because the three real datasets do not embed all evolution types.

Observations. Regarding the size of each implementation, we observe in Table 7 that Social Experiment implementation has the smallest database instance size while Citibike has the highest one. Regarding the number of nodes and edges, we observe that E-commerce implementation is composed of the highest

Implementation	Social Experiment	E-commerce	Citibike
Size (in GB)	0,3	3,6	6,7

Table 7: Size of graph database instances in Neo4j based on real datasets.

number of nodes (Table 8) while Citibike implementation is composed of the highest number of edges (Table 9). Regarding the average execution time of queries involving entities (SE) (Figure 14), the Social experiment implementation records instantaneous one. On the contrary, the average execution time of queries on entities for the E-commerce implementation explodes (>30s). No queries on entities were run on the Citibike implementation. Finally, regarding the average execution time of queries involving relationships (SU) (Figure 15), we observe that it is globally low (at most 2s) that for the three implementations. Citibike implementation records the higher average execution time of queries involving relationships.

Discussion. Regarding queries on entities (SE) (Figure 14), the gap of execution times between the E-commerce and Social experiment implementations is partly due to the difference in the number of nodes involved in queries. As queries on entities involve conditions on nodes, they involve a higher number of nodes during processing for the E-commerce implementation than the Social experiment implementation. So they require more execution times to process for the E-commerce implementation. The Social experiment implementation reduces by 99% the average execution time (of SE queries) of the E-commerce implementation. Proportionally, Social experiment implementation has 99% less nodes than the E-commerce implementation.

Regarding queries on relationships (Figure 15), the gap of execution times between the three implementations is partly due to the difference in the number of edges involved in queries. As queries on relationships involve conditions on edges, they involve a higher number of edges during processing for the Citibike

Implementation	Social Experiment	E-commerce	Citibike
Scale factor	2	3	1
Number of nodes	33 934	4 821 694	2 861

Table 8: Number of nodes and scale factors of graph database instances in Neo4j based on real datasets.

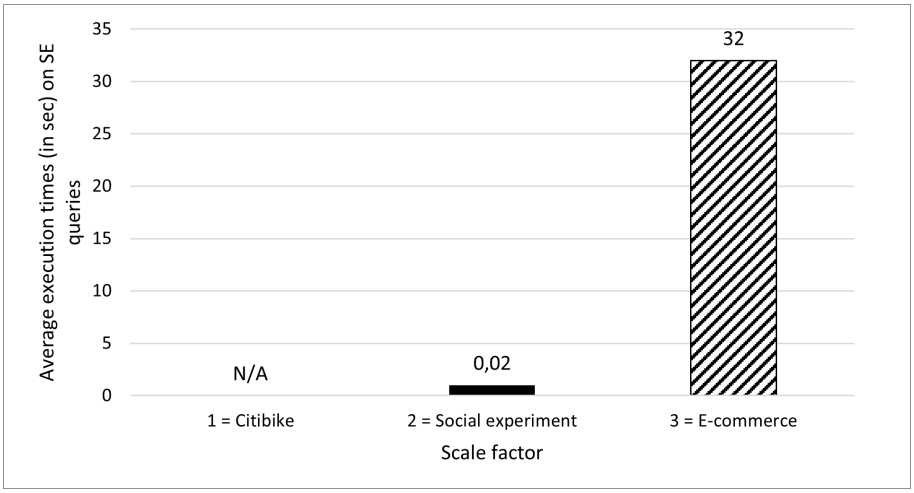


Figure 14: Average execution times of SE queries according to three scale factors. SE= Single Entity.

implementation than the Social experiment and E-commerce implementations. So they require more execution times to process for the Citibike implementation. The Social experiment implementation has 92% less edges than the Citibike implementation. It saves 80% of the average execution time (of SU queries) of the Citibike implementation. The E-commerce implementation has 81% less edges than the Citibike implementation. It saves 26% of the average execution time (of SU queries) of the Citibike implementation.

Implications. Query execution times do not depend directly on the size of the implementation but specifically on the number of nodes and edges implemented

Implementation	Social Experiment	E-commerce	Citibike
Scale factor	1	2	3
Number of edges	2 168 270	5 222 996	27 561 618

Table 9: Number of edges and scale factors of graph database instances in Neo4j based on real datasets.

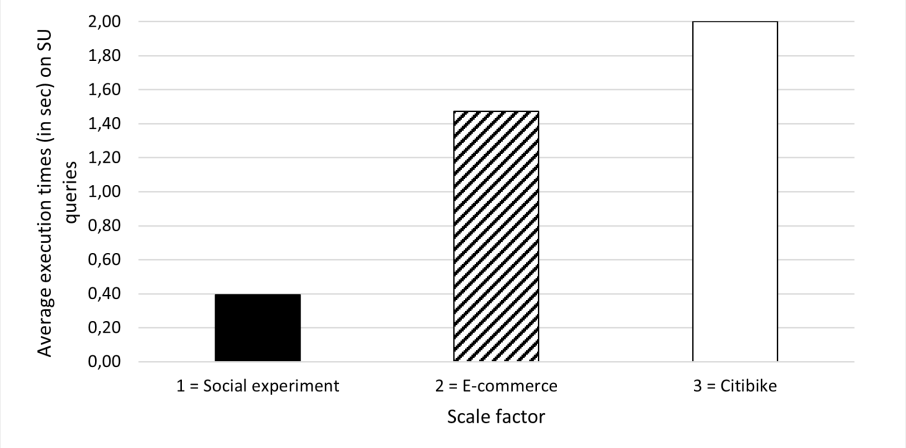


Figure 15: Average execution times of SU queries according to three scale factors. SU= Subgraph.

in Neo4j. Indeed, query execution times explode with the increase in the number of nodes while stay quite low with the increase in the number of edges (i.e. the interconnectivity) in a dataset. As Neo4j is a graph-based data store, queries involving conditions on edges are more scalable compared to queries involving conditions on nodes [38].

7. Conclusion and future work

This paper has presented a complete solution to manage temporal graph data. The power of our solution lies on the proposition of a conceptual modelling, translation rules of the latter for its implementation, and experimental

assessments to illustrate its feasibility, usability, efficiency and scalability.

We proposed a conceptual modelling of temporal graphs to represent graph data that evolve over time. The advantage of our model compared to existing approaches is first to be business-oriented. It provides to non-expert user a comprehensive overview of data and their changing components. Second, it is generic in terms of representing different types of changes of graph data: topology, attribute set and attribute value. Finally, it represents the temporal evolution of data without losing information and redundancy contrary to snapshot-based models.

To use our conceptual model in real business analyses, it must be transformed into a logical model before being implemented in a specific technical environment. To do so, we proposed standard translation rules between our model and the property graph, which is commonly used in graph-oriented NoSQL store. The advantage of our translation rules is that our model is directly convertible into the property graph without any specific developments. We verified the feasibility of our model by implementing an example dataset using our translation rules in Neo4j. Then, we verified its usability by running business analyses on evolution aspects.

To highlight the efficiency of our model, we made a comparative study of its implementation in Neo4j with the traditional sequence of snapshots and an optimized version of snapshots based on the same dataset. We observed that our model performs better than the sequence of snapshots by reducing 12 times disk usage and by saving up to 99% of query execution times. In comparison to the optimized sequence of snapshots, our model reduces 9 times disk usage and saves until 99% of query execution times. In a nusthell, our model is an efficient solution for storing and querying a dataset with temporal evolution.

To evaluate the scalability of our model, we made a comparative study of three temporal graph implementations in Neo4j based on three real-world datasets with different scales. We observe that execution times of queries involving mainly conditions on nodes explode (>30s) when the number of nodes increases. Conversely, execution times of queries mainly involving conditions on

edges stay low (at most 2s) when the number of edges increases. In short, our model has a better scalability for queries involving conditions on edges in Neo4j.

In our analyses, we have made data-oriented restitution but we can make restitution oriented to changes: for instance, if the decision maker wants to visualize the changes of a specific component in terms of attributes. We are currently working on an exploration tool of our temporal graph with a graphical interface. This first prototype includes functionalities to visualize the different evolution types in our model. Indeed, we have seen that Neo4j does not provide an expressive visualization of our concepts. It is due to the fact that Neo4j is initially designed for static graphs. It is also the case for other commercialized graph data stores. In parallel, we are working on the proposition of algebraic operators to make the technical complexity transparent to users. These operators have been proposed in an international paper in progress. The next step is to make a survey to identify (i) the Artificial Intelligence (AI) algorithms that are directly compatible with our temporal graph model and (ii) the required extensions for non-compatible AI algorithms.

8. Funding acknowledgment

This work was supported by Activus Group (https://www.activus-group.fr/), IRIT (https://www.irit.fr/) and ANRT (http://www.anrt.asso.fr/fr) with the reference number 2019/0969.

References

[1] V. C. Storey, I.-Y. Song, Big data technologies and management: What conceptual modeling can do, Data & Knowledge Engineering 108 (2017) 50–67.

[2] R. Angles, C. Gutierrez, An introduction to Graph Data Management, arXiv:1801.00036 [cs] (2018) 1–32ArXiv: 1801.00036. doi:10.1007/978-3-319-96193-4_1.

[3] B. Bbel, J. Eder, C. Koncilia, T. Morzy, R. Wrembel, Creation and management of versions in multiversion data warehouse, in: Proceedings of the 2004 ACM symposium on Applied computing - SAC '04, ACM Press, Nicosia, Cyprus, 2004, p. 717. doi:10.1145/967900.968049.

[4] P. Holme, J. Saramäki, Temporal networks, Physics reports 519 (3) (2012) 97–125, publisher: Elsevier.

[5] M. H. Böhlen, A. Dignös, J. Gamper, C. S. Jensen, Temporal Data Management – An Overview, in: E. Zimányi (Ed.), Business Intelligence and Big Data, Vol. 324, Springer International Publishing, Cham, 2018, pp. 51–83, series Title: Lecture Notes in Business Information Processing. doi:10.1007/978-3-319-96655-7_3.

[6] T. Johnston, R. Weis, A Brief History of Temporal Data Management, in: Managing Time in Relational Databases, Elsevier, 2010, pp. 11–25. doi:10.1016/B978-0-12-375041-9.00001-7.

[7] M. Bohlen, R. Busatto, C. Jensen, Point-versus interval-based temporal data models, in: Proceedings 14th International Conference on Data Engineering, IEEE Comput. Soc, Orlando, FL, USA, 1998, pp. 192–200. doi:10.1109/ICDE.1998.655777.

[8] A. Zaki, M. Attia, D. Hegazy, S. Amin, Comprehensive Survey on Dynamic Graph Models, International Journal of Advanced Computer Science and Applications 7 (2). doi:10.14569/IJACSA.2016.070273.

[9] A. Debrouvier, E. Parodi, M. Perazzo, V. Soliani, A. Vaisman, A model and query language for temporal graph databases, The VLDB Journal 30 (5) (2021) 825–858. doi:10.1007/s00778-021-00675-4.

[10] V. Z. Moffitt, J. Stoyanovich, Towards sequenced semantics for evolving graphs., in: EDBT, 2017, pp. 446–449.

[11] A. Kosmatopoulos, K. Giannakopoulou, A. N. Papadopoulos, K. Tsichlas, An Overview of Methods for Handling Evolving Graph Sequences, in:

I. Karydis, S. Sioutas, P. Triantafillou, D. Tsoumakos (Eds.), Algorithmic Aspects of Cloud Computing, Vol. 9511, Springer International Publishing, 2016, pp. 181–192. doi:10.1007/978-3-319-29919-8_14.

[12] C. S. Jensen, J. Clifford, S. K. Gadia, A. Segev, R. T. Snodgrass, A glossary of temporal database concepts, ACM SIGMOD Record 21 (3) (1992) 35–43. doi:10.1145/140979.140996.

[13] S. Ji, S. Pan, E. Cambria, P. Marttinen, P. S. Yu, A Survey on Knowledge Graphs: Representation, Acquisition and Applications, IEEE Transactions on Neural Networks and Learning Systems (2021) 1–21ArXiv: 2002.00388. doi:10.1109/TNNLS.2021.3070843.

[14] Y. Yang, J. X. Yu, H. Gao, J. Pei, J. Li, Mining most frequently changing component in evolving graphs, World Wide Web 17 (3) (2014) 351–376.

[15] R. A. Rossi, B. Gallagher, J. Neville, K. Henderson, Modeling dynamic behavior in large evolving graphs, in: Proceedings of the sixth ACM international conference on Web search and data mining - WSDM '13, ACM Press, 2013, pp. 667–676.

[16] C. Aslay, M. A. U. Nasir, G. De Francisci Morales, A. Gionis, Mining Frequent Patterns in Evolving Graphs, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, ACM, 2018, pp. 923–932.

[17] M. Latapy, T. Viard, C. Magnien, Stream Graphs and Link Streams for the Modeling of Interactions over Time, Social Networks Analysis and Mining 8 (1) (2018) 61:1–61:29. doi:10.1007/s13278-018-0537-7.

[18] E. Desmier, M. Plantevit, C. Robardet, J.-F. Boulicaut, Cohesive co-evolution patterns in dynamic attributed graphs, in: International Conference on Discovery Science, Springer, 2012, pp. 110–124.

[19] A. Zhao, G. Liu, B. Zheng, Y. Zhao, K. Zheng, Temporal paths discovery with multiple constraints in attributed dynamic graphs, World Wide Web 23 (1) (2020) 313–336. doi:10.1007/s11280-019-00670-4.

[20] A. Campos, J. Mozzino, A. Vaisman, Towards Temporal Graph Databases, arXiv:1604.08568 [cs]ArXiv: 1604.08568.
URL http://arxiv.org/abs/1604.08568

[21] L. Xiangyu, L. Yingxiao, G. Xiaolin, Y. Zhenhua, An Efficient Snapshot Strategy for Dynamic Graph Storage Systems to Support Historical Queries, IEEE Access 8 (2020) 90838–90846. doi:10.1109/ACCESS.2020.2994242.

[22] U. Khurana, A. Deshpande, Efficient snapshot retrieval over historical graph data, in: 2013 IEEE 29th International Conference on Data Engineering (ICDE), IEEE, 2013, pp. 997–1008. doi:10.1109/ICDE.2013.6544892.

[23] C. Ren, E. Lo, B. Kao, X. Zhu, R. Cheng, On querying historical evolving graph sequences, Proceedings of the VLDB Endowment 4 (11) (2011) 726–737.

[24] S. Gandhi, Y. Simmhan, An Interval-centric Model for Distributed Computing over Temporal Graphs, in: 2020 IEEE 36th International Conference on Data Engineering (ICDE), IEEE, 2020, pp. 1129–1140. doi:10.1109/ICDE48307.2020.00102.

[25] S. Ramesh, A. Baranawal, Y. Simmhan, A Distributed Path Query Engine for Temporal Property Graphs, in: 2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID), IEEE, 2020, pp. 499–508. doi:10.1109/CCGrid49817.2020.00-43.

[26] U. Khurana, A. Deshpande, Storing and Analyzing Historical Graph Data at Scale, in: EDBT, 2016.

[27] C. Cattuto, M. Quaggiotto, A. Panisson, A. Averbuch, Time-varying social networks in a graph database: a Neo4j use case, in: First International Workshop on Graph Data Management Experiences and Systems, GRADES '13, Association for Computing Machinery, 2013, pp. 1–6. doi:10.1145/2484425.2484442.

[28] H. Huang, J. Song, X. Lin, S. Ma, J. Huai, TGraph: A Temporal Graph Data Management System, in: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM, 2016, pp. 2469–2472.

[29] Y. Roussakis, I. Chrysakis, K. Stefanidis, G. Flouris, Y. Stavrakas, A flexible framework for understanding the dynamics of evolving RDF datasets, in: International Semantic Web Conference, Springer, 2015, pp. 495–512.

[30] N. Pernelle, F. Saïs, D. Mercier, S. Thuraisamy, RDF data evolution: efficient detection and semantic representation of changes, Semantic Systems-SEMANTiCS2016 (2016) 4.

[31] F. Ravat, J. Song, O. Teste, C. Trojahn, Improving the performance of querying multidimensional RDF data using aggregates, in: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC '19, Association for Computing Machinery, 2019, pp. 2275–2284.

[32] X. S. Wang, S. Jajodia, V. Subrahmanian, Temporal modules: An approach toward federated temporal databases, in: Proceedings of the 1993 ACM SIGMOD international conference on Management of data, 1993, pp. 227–236.

[33] J. F. Allen, Maintaining knowledge about temporal intervals, Communications of the ACM 26 (11) (1983) 832–843. doi:10.1145/182.358434.

[34] L. Andriamampianina, F. Ravat, J. Song, N. Vallès-Parlangeau, Towards an Efficient Approach to Manage Graph Data Evolution: Conceptual Modelling and Experimental Assessments, in: S. Cherfi, A. Perini, S. Nurcan

(Eds.), Research Challenges in Information Science, Springer International Publishing, 2021, pp. 471–488.

[35] R. Angles, The Property Graph Database Model, in: AMW, 2018.

[36] A. Madan, M. Cebrian, S. Moturu, K. Farrahi, A. S. Pentland, Sensing the "Health State" of a Community, IEEE Pervasive Computing 11 (4) (2012) 36–45. doi:10.1109/MPRV.2011.79.

[37] G. Koloniari, D. Souravlias, E. Pitoura, On Graph Deltas for Historical Queries, CoRR abs/1302.5549.

[38] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, D. Wilkins, A comparison of a graph database and a relational database: a data provenance perspective, in: Proceedings of the 48th Annual Southeast Regional Conference on - ACM SE '10, ACM Press, 2010, p. 1. doi:10.1145/1900008.1900067.