IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

Bitemporal Property Graphs: Dealing with Both Valid and Transaction Time

This is the author's manuscript

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/2094991 since 2026-04-17T14:07:29Z

Publisher:

Springer

Published version:

DOI:10.1007/978-3-032-05281-0_15

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

18 November 2026

Bitemporal Property Graphs: Dealing with Both Valid and Transaction Time

Luca Anselma[1[0000-0003-2292-6480]], Marco Ballerini[1], Paolo Giordano[1[0009-0006-3934-0426]], Erica Raina[2,3[0009-0001-4617-4992]] and Paolo Terenziani[2,3[0000-0002-9014-7537]]

[1] Dipartimento di Informatica, Università degli Studi di Torino, Turin, Italy
luca.anselma@unito.it, marco.ballerini@edu.unito.it, paolo.giordano494@edu.unito.it
[2] DISIT, University of Eastern Piedmont, Alessandria, Italy
{erica.raina, paolo.terenziani}@uniupo.it
[3] Integrated Laboratory of AI and Medical Informatics DAIRI + DISIT, Alessandria, Italy

Abstract. The need to deal with both *valid* and *transaction* time in temporal databases has been identified already in the 1980s, and bitemporal databases are fully consolidated in the relational context. However, only few graph databases deal with both times. In this paper, we extend the recent graph database AeonG to consider both transaction and valid time. We propose an extended data model, and demonstrate its expressiveness showing how it copes with a benchmark example in the literature. We extend the query language, and experimentally prove that, despite the increased expressiveness, our framework adds only a limited overhead to AeonG.

Keywords: Temporal graph databases, Valid Time, Transaction Time, Query Language, Experimental Evaluation

1 Introduction

Already in 1986, in their seminal paper [1], Snodgrass and Ahn clearly pointed out the necessity of distinguishing between ***valid time***, i.e., *the time when the information is valid* in the modelled mini-world, and ***transaction time***, i.e., *the time the information was stored in the database*. Since then, such a distinction has pervaded the area of temporal DB. For instance, in the recent Encyclopedia of Database Systems [2], the following definitions of valid and transaction time are reported.

"The valid time of a fact is the time when the fact is true in the modelled reality. Any subset of the time domain may be associated with a fact." [3]

"A database fact is stored in a database at some point in time, and after it is stored, it remains current, or part of the current database state, until it is logically deleted. The transaction time of a database fact is the time when the fact is current in the database. As a consequence, the transaction time of a fact is generally not a time instant, but rather has duration. The transaction time of a fact cannot extend into the future. Also, as it is impossible to change the past, meaning that (past) transaction times cannot be changed." [4]

Both valid and transaction time are included in TSQL2 [5], a milestone in the area of temporal relational DBs, ATSQL [6], and SQL standard (i.e., system-versioned tables for transaction time and application-time period tables for valid time in SQL:2023 [7]).

Already the seminal paper [1] reports several motivations for considering both valid and transaction time, such as support to historical queries about the past (e.g., for trend analysis), need to support retroactive and proactive changes, need to support error corrections while preserving the possibility to access past versions of the Database (e.g., for legal purposes). As an introductory running example, we adopt one of the examples provided in [1] slightly modified to include additional aspects. In Section 4 we will provide an additional example to consider also other aspects of graph databases.

Example 1.

A. Merrie joins as an instructor. Inserted in June 1973, valid from September 1973
B. Merrie is promoted to full professor. Inserted in December 1973, valid from December 1973
C. Correction: Merrie was promoted to assistant professor. Corrected in February 1974, valid from December 1973
D. Merrie is promoted to full professor. Inserted in December 1978, valid from June 1978.

Example A-D above shows that valid time (VT) and transaction time (TT) are orthogonal. In particular:

- B shows an example in which TT and VT are the same
- A shows an example of proactive change: a piece of information is inserted into the DB before it starts to hold in the modelled world
- D shows an example of retroactive change: a piece of information is inserted into the DB after it has already started to hold in the modelled world.

Example C shows a correction. Keeping track of the past states of the DB may be important. For instance, between December 1973 and February 1974 the DB contained the information that Merrie was a Full Professor. If such information is not retained (as a past version of the DB), administrative stuff may occur in legal problems, since they paid Merrie as a Full Professor, while she was not (in the real world).

The approach in [8] provides a very interesting framework to cope with temporal Graph DB. In particular, AeonG is a graph database with a built-in mono-temporal support. AeonG relies upon two physically isolated storage systems, one based on RocksDB [9], a persistent key-value store for fast storage environments, and one on Memgraph [10], a graph DB that provides resiliency, low latency and multi-tenancy. AeonG, for the sake of efficiency, proposes the “delta+anchor”: it periodically creates a complete version (namely anchor) of a graph object and maintains every change (namely delta) between two adjacent anchors of the same object. The deltas from the most recent anchor are stored in main memory through Memgraph, while anchors and past deltas are stored in secondary memory in RocksDB. AeonG has been shown to be

extremely fast, with only a 9.74% performance degradation for supporting temporal features.

A major limitation of such an approach is that it only copes with one type of time. Indeed, though the paper does not clearly specify that, we infer that it is transaction time from the fact that the starting time associated with each vertex, edge and property starts at the exact time when they are inserted in the DB (and ends when they are modified or deleted). An additional indication in favour of the "transaction time interpretation" is the fact that the start and end times cannot be inserted by users, but are automatically managed by the DBMS (which is exactly the behaviour adopted for transaction time; see, e.g., TSQL2 [5]).

The goal of our work is to extend the approach by [8] to also deal with valid time. In Section 2, we extend AeonG's definition of Temporal Property Graph to consider both valid and transaction time, and exemplify the expressiveness of our data model considering Example 1 above. In Section 3, we briefly describe our extended query language. Since Example 1 is in the tradition of temporal relational DBs literature (and a single node suffices to cope with it), in Section 4 we propose an additional example, to further stress and clarify the impact of our approach in the Temporal Graph context. In Section 5 we describe our prototypical implementation, and experimentally demonstrate that, despite the increased expressiveness, our approach only adds a limited overhead to AeonG's one. Finally, Section 6 contains conclusions, related and future work.

2 Bitemporal Property Graphs

In this Section, we propose our data model. We define Bitemporal Property Graphs, describe the temporal constraints we impose, and show their expressiveness through its application to Example 1 above.

2.1 Data Model

The starting point of our approach is the definition of the domains of time. As in BCDM (Bitemporal Conceptual Data Model) [11] (a unifying data model, isolating the "core" semantics underlying many temporal relational approaches, including TSQL2 [5]) we consider both *valid time* and *transaction time*. Both time domains are totally ordered and isomorphic to the subsets of the domain of natural numbers.

Our definition of BiTemporal Property Graphs is an extension of the definition in [8], to include also transaction time.

Definition 1 (BiTemporal Property Graph). Let $\mathcal{N}$ and $\mathcal{E}$ denote sets of vertex ids and edge ids, and $\mathcal{L}$, $\mathcal{K}$ and $\mathcal{V}$ be countable sets of labels, property names and property values. A BiTemporal Property Graph is a tuple $G = \langle \Omega^{vt}, \Omega^{tt}, N, E, \rho, \lambda, \pi, \sigma, \tau \rangle$ where:

- Ω^{vt} is a temporal domain, which is a finite set of consecutive time points, that is, $\Omega^{vt} = \{i \in T^{v} \mid a \leq i \leq b\} \cup \{\infty\}$ for some $a, b \in T^{v}$ such that $a \leq b$. T^{v} represents the universe of valid-time points;

- Ω^{tt} is a temporal domain, which is a finite set of consecutive time points, that is, $\Omega^{tt} = \{i \in T^t \mid a \leq i \leq b\} \cup \{\infty\}$ for some a, $b \in T^t$ such that $a \leq b$. T^t represents the universe of transaction-time points;
- N is a finite subset of $\mathcal{N}$, whose elements are referred to as the vertices of G;
- E is a finite subset of $\mathcal{E}$, whose elements are referred to as the edges of G and $N \cap E = \emptyset$;
- ρ: $E \rightarrow (N \times N)$ is a total function mapping each edge to its source and destination vertices;
- λ: $(N \cup E) \rightarrow 2^{\mathcal{L}}$ is a total function mapping vertices and edges to finite sets of labels (including the empty set);
- $\sigma^{vt,tt}$: $(N \cup E) \times \Omega^{vt} \times \Omega^{tt} \rightarrow \{true, false\}$ is a total function that maps a vertex or an edge and a bitemporal point to a Boolean variable, indicating whether this vertex or edge holds at that bitemporal point. Notably, a vertex or an edge holds in a bitemporal point $<c^{vt},c^{tt}> \in \Omega^{vt} \times \Omega^{tt}$ if it holds in the modelled world at $c \in \Omega^{vt}$, and it exists (is logically present) in the DB at $c^t \in \Omega^{tt}$.

τ: $(N \cup E) \times \mathcal{K} \times \Omega^{vt} \times \Omega^{vt} \times \Omega^{tt} \times \Omega^{tt} \rightarrow \mathcal{V}$ is a partial function that maps a vertex or an edge and a property key, a time period ω^{vt} for the valid time and a period ω^{tt} for the transaction time to a value.

Notably, though not explicitly introduced in the definition of Temporal Property Graph, the approach in [8] also adopts a special value, "∞", to denote the fact that the given entity (node or event) holds in the DB until it will possibly be changed/removed.

Terminology. In the following and above, we call ***bitemporal point*** each pair of time points $<c^v,c^t> \in \Omega^{vt} \times \Omega^{tt}$.

Note. We assume that intervals are closed to the left and open to the right.

Constraints. In our bitemporal property graph model, we impose two constraints to enforce that the graph at any time point corresponds to a valid property graph. At any ***transaction*** time point $c^t \in \Omega^{tt}$:

(1) An edge exists in the DB only if both source and destination vertices exist in the DB at c^t. Formally, if $e \in E$ and $\exists c^v \in \Omega^{vt}$ $\sigma(e, c^v, c^t) = true$ and $\rho(e) = (v1, v2)$, then $\exists c^{v'} \in \Omega^{vt}$ $\sigma(v1, c^{v'}, c^t) = true$ and $\exists c^{v''} \in \Omega^{vt}$ $\sigma(v2, c^{v''}, c^t) = true$;

(2) A property can only take on a value during the bitemporal time period when the corresponding vertex or edge exists. Formally, if $\tau(o, k, c^{vs}, c^{ve}, c^{ts}, c^{te}) = val$, where $o \in (N \cup E)$, $k \in \mathcal{K}$, $val \in \mathcal{V}$, then $\forall c^t \in [c^{ts}, c^{te})$ and $\forall c^v \in [c^{vs}, c^{ve})$ we have $\sigma(o, c^v, c^t) = true$.

Note. As a remarkable side-effect, the deletion of a node implies the deletion of all its entering and exiting edges. Notably, no piece of information is physically deleted from the bitemporal graph. The deletion of an entity e with transaction time $[c^{ts}, +\infty)$ at a transaction time $c^t \in \Omega^{tt}$ is achieved by setting its transaction time to $[c^{ts}, c^t)$.

Intuitively, temporal constraints in [8] impose that properties and relationships of any vertex are "synchronous" with the vertex (i.e., they may hold only during the time

of the vertex). Such constraints are reasonable when considering transaction time: if a vertex v is inserted in the Graph at time s and is deleted at e (i.e., its transaction time is [s,e)), it cannot have properties/relationships in the graph out of such a period.

However, such a constraint would be too restrictive when applied to valid time, since many phenomena in the "real" world would not be captured by the graph DB.

Example 2. Suppose, e.g., that one wants to model family trees, considering a property (or a relationship) Great-Grandfather. Obviously, the great-grandfather's life is not necessarily contained in the life of the great-grandson.

Example 3. As a further example, suppose that one wants to model in a graphDB the notion of *causation*, modelling it through an edge connecting two vertices, representing cause and effect. Temporal constraints between causes and effects are quite various, and have been widely studied (see, e.g., [12]), but in general effects start after the beginning of their causes (or even after their end, as in "one-shot" causation). An example concerning causation is proposed in Section 4.

2.2 An Example: The Evolution of Merrie's Rank

As an example, we consider Example 1 in the introduction. We assume the granularity of months both for valid and transaction time, and we take January 1973 as the origin of time (so "1" is the month of January 1973, "2" is February 1973 and so on).

Independently of the temporal issues, there are (at least) two different ways to model the example: (i) considering a node for employees, and representing ranks as an attribute, or (ii) considering a node for employees, one for ranks, and an edge to connect them. In the following, for the sake of brevity, we chose the first alternative, but also the second can be modelled in our approach.

A. Merrie joins as an instructor. Inserted in June 1973, valid from September 1973

For the temporal attributes, we represent them as a pair [Valid time start, Valid time end) followed by a pair [Transaction time start, Transaction time end).

V0: Employee [6, ∞)[6, ∞)
V01: Name: Merrie [6, ∞)[6, ∞)
V02: Rank: Instructor [9, ∞)[6, ∞)

The time point 6 (both in valid time and transaction time) denotes June 1973. The node employee, with attribute *Name* with value "*Merrie*" is inserted (transaction time) at 6, and starts to hold at 6 (valid time). However, the attribute *Rank* assumes its value at time 9.

B. Merrie is promoted to full professor. Inserted in December 1973, valid from December 1973

V0: Employee [6, ∞)[6, ∞)
V01: Name: Merrie [6, ∞)[6, ∞)
V02: Rank: Instructor [9, ∞)[6, 12)
V03: Rank: Instructor [9, 12)[12, ∞)
V04: Rank: Full Professor [12, ∞)[12, ∞)

From transaction time 6 to 12 (excluded) the DB contained the information that Merrie was an instructor (from 9 on). Such information is retained. However, starting from transaction time 12, the DB has the information that Merrie was an Instructor from 9 to 12 (excluded), and is a Full Professor since 12.

C. Correction: Merrie was promoted to assistant professor. Corrected in February 1974, valid from December 1973

V0: Employee [6, ∞)[6, ∞)
V01: Name: Merrie [6, ∞)[6, ∞)
V02: Rank: Instructor [9, ∞)[6, 12)
V03: Rank: Instructor [9, 12)[12, ∞)
V04: Rank: Full Professor [12, ∞)[12, 14)
V05: Rank: Assoc Professor [12, ∞)[14, ∞)

Until time 14, the DB contains the information described at step B above. At time 14, the information that Merrie is a Full Professor is logically deleted (by setting to 14 the upper bound of transaction time), while it is added (lower bound of transaction time) the information that, starting from month 12 (valid time [12, ∞)), Merrie is an Associate Professor.

D. Merrie is promoted to full professor. Inserted in December 1978, valid from June 1978

V0: Employee [6, ∞)[6, ∞)
V01: Name: Merrie [6, ∞)[6, ∞)
V02: Rank: Instructor [9, ∞)[6, 12)
V03: Rank: Instructor [9, 12)[12, ∞)
V04: Rank: Full Professor [12, ∞)[12, 14)
V05: Rank: Assoc Professor [12, ∞)[14, 72)
V06: Rank: Assoc Professor [12, 66)[72, ∞)
V07: Rank: Full Professor [66, ∞)[72, ∞)

Until time 72, the DB contains the information described at step C above. At (transaction) time 72, we add the information that Merrie ended to be Associate Professor at (valid) time 66, and since that time she is a Full Professor.

Comment (how to read). At each snapshot (A, A+B, A+B+C, A+B+C+D) of the evolution of the DB, the "current status" can be obtained by considering only those properties whose right bound of the transaction time is "∞".

For example, in the snapshot A+B+C the "current situation" was:

V0: Employee [6, ∞)[6, ∞)
V01: Name: Merrie [6, ∞)[6, ∞)
V03: Rank: Instructor [9, 12)[12, ∞)
V05: Rank: Assoc Professor [12, ∞)[14, ∞)

More generally, to see the DB situation at a given Transaction Time t, one simply has to consider only those properties whose transaction time period contains t. For

instance, at transaction time 13 (January 1974), it was believed that Merrie had been an Instructor from September to December 1973, and she became a Full Professor, i.e.,

V0: Employee [6, ∞)[6, ∞)
V01: Name: Merrie [6, ∞)[6, ∞)
V03: Rank: Instructor [9, 12)[12, ∞)
V04: Rank: Full Professor [12, ∞)[12, ∞)

3 Manipulation and Query Language

AeonG incorporates, as a manipulation and query language, a temporal-enhanced Cypher, which extends the standard syntax defined in OpenCypher [13] to support temporal queries. In short, AeonG introduces two temporal extensions in the MATCH clause:

- `FOR TT AS OF` t, which retrieves all graph objects legal at time t (called *time-point queries*),
- `FOR TT FROM` $t1$ `TO` $t2$, which locates all graph objects consistently legal within the time range from $t1$ to $t2$ (called *time-slice queries*).

Such limited extensions suffice, since AeonG only considers transaction time which, by definition, must be automatically managed by the DBMS (it represents the DB insertion and deletion/update time). We retain such extensions in our approach, but, in order to deal also with *valid time*, several extensions are needed:

i. extend the MATCH construct to query also the valid time,
ii. extend the *manipulation* operators to provide the possibility to specify the valid time,
iii. extend the *pattern* definition in order to be able to specify valid times for vertices, edges and properties.

Notably, while in (i) valid time is a time point or a time period, in (ii) and (iii) valid time consists of a -possibly unitary- set of non-overlapping periods (see Definition 1). In this paper, we focus only on aspects (i) and (ii), while (iii) is left as future work.

3.1 Extensions To The Query Language

In this paper, we extend the AeonG framework specifically with the purpose of expressing time-point and time-slice queries considering valid time, or transaction time, or both. We thus have, besides the cases above (in our approach, "VT" stands for valid time), also the cases:

- `FOR VT AS OF` t, which retrieves all graph objects holding in the modelled world at time t (time-point query),
- `FOR VT FROM` $t1$ `TO` $t2$, which locates all graph objects holding in the modelled world within the time range from $t1$ to $t2$ (time-slice query).

Example 4. What was Merrie's rank in January 1974 considering the info in the DB in January 1974?

```
MATCH (n:Employee {Name: Merrie}) FOR TT AS OF 13 FOR VT AS OF 13 RETURN n.
```

- Full Professor [12, ∞)[12, ∞)

Example 5. What was Merrie's rank in January 1974 considering the info in the DB from January 1974 to March 1974?

```
MATCH (n:Employee {Name: Merrie}) FOR TT FROM 13 TO 15 FOR VT AS OF 13 RETURN n.
```

- Full Professor [12, ∞)[12, ∞)
- Full Professor [12, ∞)[12, 14)
- Assoc Professor [12, ∞)[14, ∞)

Example 6. What was Merrie's rank from October 1973 to January 1974 considering the info in the DB from January 1974 to March 1974?

```
MATCH (n:Employee {Name: Merrie}) FOR TT FROM 13 TO 15 FOR VT FROM 10 TO 13 RETURN n.
```

- Instructor [9, 12)[12, +∞)
- Full Professor [12, +∞)[12, +∞)
- Full Professor [12, +∞)[12, 14)
- Assoc Professor [12, +∞)[14, +∞)

3.2 Manipulation Operators

To illustrate our extensions to manipulation operation, we re-consider Example 1. Results obtained by the application of the operations below are the ones already presented in Section 2.

A. Merrie joins as an instructor. Inserted in June 1973, valid from September 1973

At transaction time 6, `CREATE (:Employee {Name: Merrie}) FOR VT FROM 6 TO "+INF"`.
At transaction time 6, `MATCH (n:Employee {Name: Merrie}) FOR VT FROM 9 TO "+INF" SET n.Rank='Instructor'`.

B. Merrie is promoted to full professor. Inserted in December 1973, valid from December 1973

At transaction time 12, `MATCH (n:Employee {Name: Merrie}) FOR VT FROM 9 TO 12 SET n.Rank='Instructor'`.

At transaction time 12, `MATCH (n:Employee {Name: Merrie}) FOR VT FROM 12 TO "+INF" SET n.Rank='Full Professor'`.

C. Correction: Merrie was promoted to assistant professor. Corrected in February 1974, valid from December 1973

At transaction time 14, `MATCH (n:Employee { Name : Merrie}) FOR VT FROM 12 TO "+INF" SET n.Rank = 'Assoc Professor'.`

D. Merrie is promoted to full professor. Inserted in December 1978, valid from June 1978

At transaction time 72, `MATCH (n:Employee { Name : Merrie}) FOR VT FROM 12 TO 66 SET n.Rank = 'Assoc Professor'.`

At transaction time 72, `MATCH (n:Employee { Name : Merrie}) FOR VT FROM 66 TO "+INF" SET n.Rank = 'Full Professor'.`

4 A Further Example: Nodes & Edges

Example 1 above is one of the most famous examples in the tradition of temporal relational DBs, to show the need of coping with both valid time and transaction time. However, Example 1 can be "naturally" modelled by a one-node graph, so that one may wonder whether the same concepts are also important in the context of graphs. In the following we propose a small additional example consisting of three nodes and two edges. Though "minimal", such an example allows us to point out important concepts concerning the need to manage both valid and transaction times also in graphs.

Our example is in the domain of medical informatics since, though our bitemporal graph DB approach is completely domain-independent, we intend to apply our approach within such a context. Indeed, our group has an almost thirty-year tradition in providing AI and temporal DB supports to clinical decision support and medical educational systems, within the long-term Guideline Representation, Acquisition and Execution (GLARE) project [14–19].

Example 7. Let us model a DB of therapies, which are administered to patients and produce effects, consisting of the following facts:

7.a Anna (born on 1980-03-03) followed the Corticosteroid therapy from 2023-01-10 to 2023-01-20 (included). The therapy caused an elevated intraocular pressure from 2023-02-01 to 2023-02-15 (notice that the effect starts after the end of the therapy). Such information was stored in the DB on 2023-03-01.

7.b On 2023-04-01 (i.e., after 2023-03-01), an error is noticed: the effect was not elevated intraocular pressure but mild retinal edema, starting on 2023-02-03 and ending on 2023-02-17. Such an update was applied to the DB.

7.c Anna died on 2023-10-01.

7.d On 2023-11-01, the queries Q1 e Q2 were asked:

Q1 What was the effect of the Corticosteroid therapy on Anna, given the info in the DB on 2023-03-15?

Q2 What was the effect of the Corticosteroid therapy on Anna, given the info in the DB on 2023-10-15?

We can support the preceding facts with the following query statements (using dates as time points).

7.a At transaction time 2023-03-01,

- `CREATE (:Patient {Name:Anna}) FOR VT FROM "1980-03-03" TO "+INF"`
- `CREATE (:Therapy {Type:Corticosteroid}) FOR VT FROM "2023-01-10" TO "2023-01-20"`
- `CREATE (:Effect {Type:elevated intraocular pressure}) FOR VT FROM "2023-02-01" TO "2023-02-15"`
- ```
  MATCH (t:Therapy {Type:Corticosteroid}), (p:Patient {Name:Anna})
  CREATE (t)-[:Applied_to]->(p) FOR VT FROM "2023-01-10" TO "2023-01-20"
  ```
- ```
  MATCH (t:Therapy {Type:Corticosteroid}), (e:Effect {Type:elevated intraocular pressure})
  CREATE (t)-[:Has_Effect]->(e) FOR VT FROM "2023-02-01" TO "2023-02-15"
  ```

The transaction time of all the nodes and edges is [2023-03-01, +INF). On the other hand, we have different VTs. Notably, the VT of the effect "elevated intraocular pressure" (the effect of the therapy) is [2023-02-01, 2023-02-16) and is later than the VT of the corticosteroid therapy. This aspect cannot be captured in all those models that (like AeonG) dictate that edges can only exist at the time of the nodes they connect.

7.b At transaction time 2023-04-01,

- `CREATE (:Effect {Type:mild retinal edema}) FOR VT FROM "2023-02-01" TO "2023-02-15"`
- ```
  MATCH (:Therapy {Type:Corticosteroid})-[a:Has_Effect]->(:Effect {Type:elevated intraocular pressure})
  DELETE a
  ```
- ```
  MATCH (t:Therapy {Type:Corticosteroid}), (e:Effect {Type:mild retinal edema})
  CREATE (t)-[:Has_Effect]->(e) FOR VT FROM "2023-02-01" TO "2023-02-15"
  ```

The update is obtained by deleting the Has_Effect edge from Corticosteroid to Elevated intraocular pressure (i.e., its TT becomes [2023-03-01, 2023-04-01)) and inserting a new Has_Effect edge from Corticosteroid therapy to Mild retinal edema effect with TT [2023-04-01,+INF).

7.c At transaction time 2023-10-01,

- ```
  MATCH (p:Patient {Name:Anna})
  DELETE p
  ```
- `CREATE (:Patient {Name:Anna}) FOR VT FROM "1980-03-03" TO "2023-10-02"`
  ```

Notably, the death of Anna leads to an update of her VT from [1980-03-03, +INF] to [1980-03-03, 2023-10-02). Such new info is inserted with TT [2023-10-01, +INF).

Q1: What was the effect of the Corticosteroid therapy on Anna, given the info in the DB at 2023-03-15?

- `MATCH (e:Effect) <- [:Has_Effect] - (t:Therapy {Type:Corticosteroid}) - [a:Applied_to] -> (p:Patient {Name:Anna}) FOR TT AS OF "2023-03-15" RETURN e`
  `>> elevated intraocular pressure [2023-02-01, 2023-02-16)`

Q2: What was the effect of therapy Corticosteroid on Anna, given the info in the DB at 2023-10-15?

- `MATCH (e:Effect) <- [:Has_Effect] - (t:Therapy {Type: Corticosteroid}) - [a:Applied_to] -> (p:Patient {Name:Anna}) FOR TT AS OF "2023-10-15" RETURN e`
  `>> mild retinal edema [2023-02-01, 2023-02-16)`

5 Implementation and Experimental Evaluation

In this section, we sketch our implementation and describe some experimental results.

5.1 Implementation

In this section, we briefly introduce our extension of AeonG, called AeonG-VT, along with the important changes we made to AeonG's system architecture to cope with valid time.

To store multiple valid time intervals, we added to each graph object a newly created object called *VTStore*. This object stores, for each graph object property (and for the object itself), its valid time intervals using vectors of a custom-made class called *VTDateTime*.

We added three new attributes to the structure of delta:

- *applied_vt* (VTDateTime start, VTDateTime end): The valid time in the query,
- *new_value* (propertyValue): The new value of that specific property,
- *vt* (VTDateTime start, VTDateTime end): The applied valid time for the delta and, for extension, the value of valid time when a rollback is required.

Regarding the historical storage, we changed the structure of the key, adding the valid time. With this small change, we can store the valid time in the physical disk.

5.2 Experiments

We performed the experiments on a physical device with 11th Gen Intel® Core™ i7-11370H @ 3.30GHz × 8, 15.4 GiB RAM on Ubuntu 20.04.6 LTS. The garbage collector's interval for both AeonG and AeonG-VT was set to 60 seconds.

We devised two main experiments.

1. The first experiment compares the performance of a set of queries without a specific valid time, on both AeonG and AeonG-VT, with the same previously described dataset in the database.
2. The second experiment considers AeonG-VT only and evaluates its performance on valid time queries. We performed each query three times and report the average time.

5.3 Comparing AeonG-VT and AeonG

Following AeonG's paper, we have experimentally evaluated the performance of the approach considering the temporal-enhanced workload of T-mgBench, which is based on data derived from the Pokec real-world Slovak social network [20]. As the initial dataset, T-mgBench consists of 10,000 vertices and 121,716 edges. Following AeonG's approach, we added to the initial dataset a randomly generated set of updates, specifically 80,000 updates: MATCH-SET, MATCH-DELETE, and MATCH-CREATE (not considering VT). We generated the update dataset with the *create_temporal_query.py* script to compare AeonG-VT and AeonG in a realistic environment and under the same conditions. This dataset is a mix of 80% updates, 10% non-temporal creates, and 10% deletes.

In this experiment, we run the four types of queries used in AeonG's evaluation [8].

Table 1. Transaction-time queries in AeonG and AeonG-VT.

Query	Statement
Q1.1	`MATCH (n:User) FOR TT AS OF t RETURN n`
Q1.2	`MATCH (n:User) FOR TT FROM t1 TO t2 RETURN n`
Q1.3	`MATCH (n:User)-[r]->(m) FOR TT AS OF t RETURN m, n, r`
Q1.4	`MATCH (n:User)-[r]->(m) FOR TT FROM t1 TO t2 RETURN m, n, r`

After creating the database, we divided the current transaction time into ten intervals. With those intervals, we replaced the parameters in the queries, and we evaluated the performance in AeonG and AeonG-VT. In Figure 1 we show the results for queries of type Q1.4 (other results are analogous).

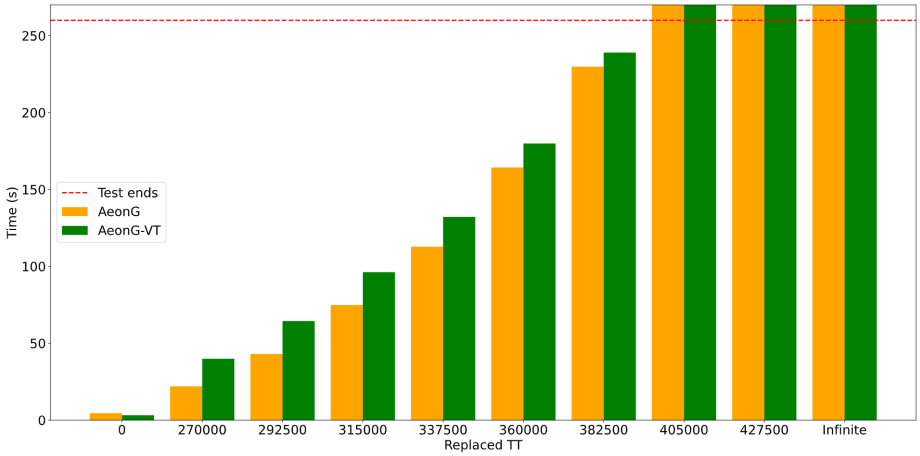


Fig. 1. Comparison of Q1.4 queries between AeonG and AeonG-VT.

The experiments clearly show that, considering TT only, despite the increased expressiveness of our approach, the overhead we added to AeonG is almost negligible.

5.4 Evaluating VT Queries in AeonG-VT

For evaluating VT queries, we created a different workload enriching the T-mgBench dataset. Our new dataset is based on the plain T-mgBench dataset and is generated by a script, to add to the vertices their valid time. This script takes two *.cypher* files: the chosen cypher dataset (in our case T-mgBench small) and the updated dataset already generated with the "create_temporal_query.py" script. The script adds to each vertex a single valid-time interval, evenly selected from a set of ten non-overlapping intervals.

The following types of VT queries are asked on the newly-created dataset.

Table 2. Valid time queries in AeonG-VT

Query	Statement
Q2.1	`MATCH (n:User) FOR VT AS OF t RETURN n`
Q2.2	`MATCH (n:User) FOR VT FROM t1 TO t2 RETURN n`
Q2.3	`MATCH (n:User)-[r]->(m) FOR VT AS OF t RETURN m, n, r`
Q2.4	`MATCH (n:User)-[r]->(m) FOR VT FROM t1 TO t2 RETURN m, n, r`

In Figure 2, we show the time needed to answer queries of type Q2.2, considering different degrees of selectivity (from 10% to 100%) of the queries. Notably, the query time increases linearly with the number of selected vertices. In all the cases, all the vertices in the dataset are selected, in less than 1 second.

In Figure 3 we show times for queries of type Q2.4, which consider vertex+edge+vertex triples. In all the cases (i.e., all the possible triples in the graph are selected), the answer time is less than 15 seconds.

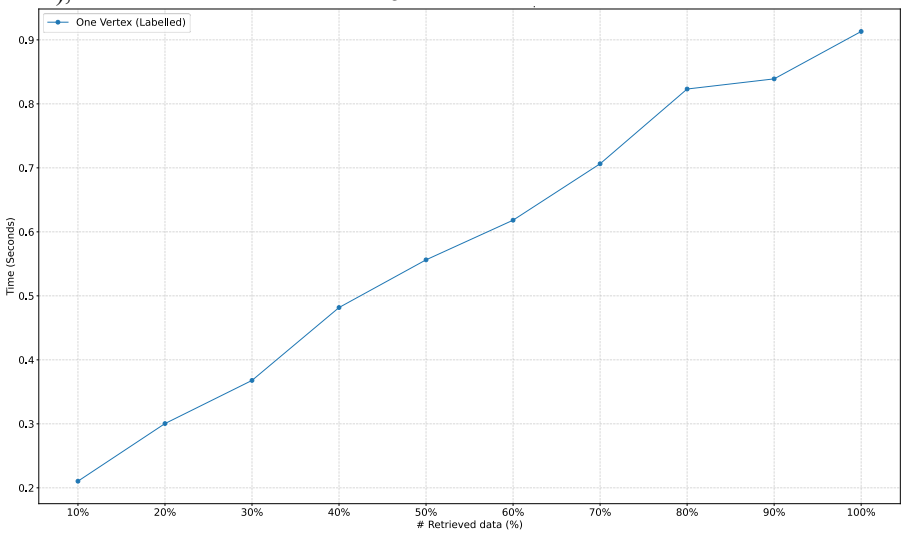


Fig. 2. Q2.2 queries performance evaluation.

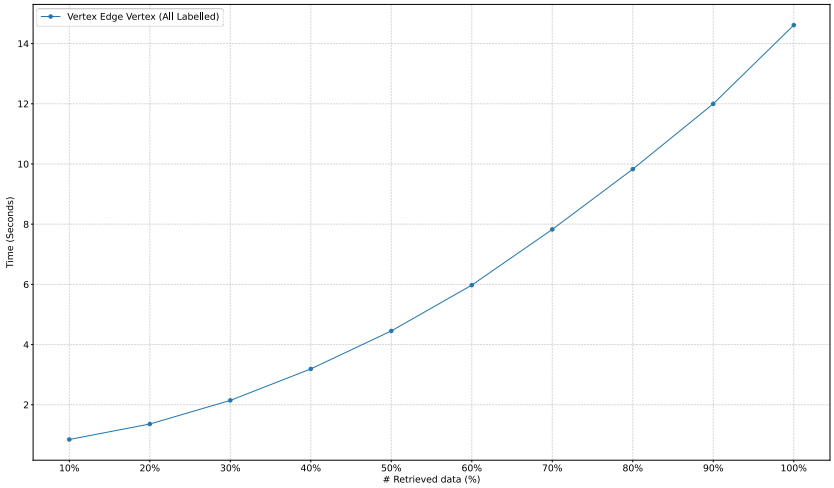


Fig. 3. Q2.4 queries performance evaluation.

6 Conclusions

Even though the treatment of both *valid* and *transaction* time has a long tradition in relational DBs, in the area of Graph DBs such a topic has only be partially explored. Indeed, to the best of our knowledge, only a few approaches considered time in the

Graph DB context, and none of them supporting two temporal dimensions (i.e., ***both*** transaction ***and*** valid time). This is a relevant drawback of the area, given the wide relevance of bitemporal data, as ascertained by 30 years of relational and object-oriented temporal DB literature. The approach in [21] deals with one temporal dimension (indeed, since an explicit definition is not provided, it is not totally clear whether the approach in [21] deals with transaction time or with valid time). However, the approach does not define a ***bitemporal*** model. The implementation is built on top of Neo4J and manages time through an external layer. Furthermore, it does not provide an exhaustive evaluation of the system, as the authors only evaluate the impact of the factors that influence the system's performance. A very promising approach to temporal DB Graph has been recently proposed by AeonG [8], considering only one temporal dimension.

In this paper we extend AeonG's data model, and demonstrate the expressiveness of our extended model showing how it copes with a benchmark example in the literature. We extend the query language of AeonG, and experimentally prove that, despite the increased expressiveness, our framework adds only a very limited overhead to AeonG.

As future work, we aim to extend the query language to specify (1) different valid times for vertices, edges and properties in the same query, and (2) qualitative temporal constraints such as, e.g., Allen's relations along the lines suggested by [21].

Acknowledgments. Erica Raina is a PhD student enrolled in the National PhD program in Artificial Intelligence, XXXIX cycle, course on Health and life sciences, organized by Università Campus Bio-Medico di Roma.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Snodgrass, Llsoo Ahn: Temporal Databases. Computer. 19, 35–42 (1986). https://doi.org/10.1109/mc.1986.1663327.
2. Liu, L., Özsu, M.T. eds: Encyclopedia of Database Systems. Springer New York, New York, NY (2018). https://doi.org/10.1007/978-1-4614-8265-9.
3. Jensen, C.S., Snodgrass, R.T.: Valid Time. In: Liu, L. and Özsu, M.T. (eds.) Encyclopedia of Database Systems. pp. 4359–4360. Springer, New York, NY (2018). https://doi.org/10.1007/978-1-4614-8265-9_1066.
4. Jensen, C.S., Snodgrass, R.T.: Transaction Time. In: Liu, L. and Özsu, M.T. (eds.) Encyclopedia of Database Systems. pp. 4200–4201. Springer, New York, NY (2018). https://doi.org/10.1007/978-1-4614-8265-9_1064.
5. Snodgrass, R.T. ed: The TSQL2 Temporal Query Language. Springer US, Boston, MA (1995). https://doi.org/10.1007/978-1-4615-2289-8.
6. Böhlen, M.H., Jensen, C.S., Snodgrass, R.T.: Temporal statement modifiers. ACM Trans. Database Syst. 25, 407–456 (2000). https://doi.org/10.1145/377674.377665.

7. ISO/IEC 9075-16:2023 Information technology — database languages — SQL — part 16: Property graph queries (SQL/PGQ). International Organization for Standardization, Geneva (2023).
8. Hou, J., Zhao, Z., Wang, Z., Lu, W., Jin, G., Wen, D., Du, X.: AeonG: An Efficient Built-in Temporal Support in Graph Databases. Proc. VLDB Endow. 17, 1515–1527 (2024). https://doi.org/10.14778/3648160.3648187.
9. RocksDB | A persistent key-value store, http://rocksdb.org/, last accessed 2025/05/06.
10. Memgraph, https://memgraph.com/, last accessed 2025/05/06.
11. Jensen, C.S., Snodgrass, R.T.: Semantics of time-varying information. Information Systems. 21, 311–352 (1996). https://doi.org/10.1016/0306-4379(96)00017-8.
12. Terenziani, P.: Towards a causal ontology coping with the temporal constraints between causes and effects. International Journal of Human-Computer Studies. 43, 847–863 (1995). https://doi.org/10.1006/ijhc.1995.1078.
13. Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., Taylor, A.: Cypher: An Evolving Query Language for Property Graphs. In: Proceedings of the 2018 International Conference on Management of Data. pp. 1433–1445. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3183713.3190657.
14. Terenziani, P., Molino, G., Torchio, M.: A modular approach for representing and executing clinical guidelines. Artif Intell Med. 23, 249–276 (2001).
15. Anselma, L., Terenziani, P., Montani, S., Bottrighi, A.: Towards a comprehensive treatment of repetitions, periodicity and temporal constraints in clinical guidelines. Artif Intell Med. 38, 171–195 (2006). https://doi.org/10.1016/j.artmed.2006.03.007.
16. Terenziani, P., German, E., Shahar, Y.: The temporal aspects of clinical guidelines. Stud Health Technol Inform. 139, 81–100 (2008).
17. Piovesan, L., Anselma, L., Terenziani, P.: Temporal detection of guideline interactions. Presented at the HEALTHINF 2015 - 8th International Conference on Health Informatics, Proceedings; Part of 8th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2015 (2015). https://doi.org/10.5220/0005186300400050.
18. Spiotta, M., Terenziani, P., Dupré, D.T.: Temporal Conformance Analysis and Explanation of Clinical Guidelines Execution: An Answer Set Programming Approach. IEEE Transactions on Knowledge and Data Engineering. 29, 2567–2580 (2017). https://doi.org/10.1109/TKDE.2017.2734084.
19. Bottrighi, A., Grosso, F., Ghiglione, M., Maconi, A., Nera, S., Piovesan, L., Raina, E., Roveta, A., Terenziani, P.: A Symbolic AI Approach to Medical Training. Journal of Medical Systems. 49, (2025). https://doi.org/10.1007/s10916-024-02139-y.
20. Pokec.sk - chat, kde je vždy najviac ľudí, https://pokec.azet.sk/, last accessed 2025/05/06.
21. Debrouvier, A., Parodi, E., Perazzo, M., Soliani, V., Vaisman, A.: A model and query language for temporal graph databases. The VLDB Journal. 30, 825–858 (2021). https://doi.org/10.1007/s00778-021-00675-4.