Transforming Time Series into Graphs and Back with HyGraph

Mouna Ammar[1,*], Shubhangi Agarwal[2], Angela Bonifati[3] and Erhard Rahm[1]

[1]Leipzig University and ScaDS.AI, Leipzig, Germany
[2]Lyon 1 University, LIRIS, Lyon, France
[3]Lyon 1 University, LIRIS and IUF, Lyon, France

Abstract

Existing graph data management systems still provide limited support for evolving and temporal data. In addition, time-series data often reside outside graph engines, hindering unified analysis. HyGraph is a new hybrid approach to manage and analyze both temporal graph and time series data in a unified manner. In particular, it supports rich transformations between graph and time-series data. We discuss two novel operators on HyGraph to illustrate such transformations, a time-series-based graph operator and a graph-based time-series operator. The first ingests time-series data and produces a new graph (or a subgraph) that captures relationships among time series based on correlation values. The second operator, in contrast, generates a time series based on the evolution of temporal graph metrics, such as aggregated edges or changes in node degree. The transformation operators allow the augmentation of derived values to the hybrid structure for self-enrichment. We also outline open challenges of dynamic transformations within the hybrid model.

Keywords

hygraph, hybrid graph, property graph, temporal graph, time-series, multi-model

1. Introduction

Graphs are a powerful tool for modeling interconnected real-world data, widely used in domains such as social networks, knowledge graphs, and urban mobility. Many of these applications inherently involve temporal dynamics, where graph elements evolve. For instance, sensor networks continuously generate time-series data [1, 2], and ride-sharing platforms track vehicle metrics over time [3, 4]. Existing graph database systems are often limited in their ability to natively manage and analyze such evolving temporal data. By contrast, the representation of time series data falls short of preserving interaction between the entities. The time series databases (TSDBs) are designed to efficiently store and analyze temporal data and are not optimized for capturing complex graph structures. They primarily focus on sequential data retrieval and aggregation [5], lacking native support for graph traversal, multi-hop, or relationship-based analytics. Further, high-dimensional time series data challenges traditional mining techniques, motivating graph-based representations as a powerful tool for analysis and visualization [6]. As a result, time-series data in graph applications is often stored separately, either in side systems or as attributes in graph databases, leading to inefficient data management.

HyGraph aims at addressing these limitations with a unified model that seamlessly integrates property graphs with time-series data. HyGraph directly represents time-dependent attributes and supports new transformation operators for evolving graph analytics. A broader discussion of HyGraph's vision and related work can be found in [7], where we also outline the motivation behind the approach and its high-level goals. In contrast, this paper provides a detailed exploration of the HyGraph data model and transformation operations, like extracting time series from graph elements and computing similarity-based transformations.

We start by formally introducing the HyGraph model in Section 2, followed by its UML and system architecture in Section 3. We present the two key transformation operators in Section 4 and illustrate the applicability of our model through a micro-mobility use case (Section 5). Broader implications and future directions are discussed in Section 6.

2. HyGraph Data Model

Analyzing data that combines graph structures and time series offers deeper insights than separate analyses. For instance, in micro-mobility applications, tracking how usage patterns evolve alongside spatial station layouts can predict demand and uncover efficient vehicle distribution strategies. Although there have been efforts to unify graph and time-series data, they often rely on graph representations for both, limiting the depth of time series analysis and relegating time series to a secondary role, primarily representing property evolution [8, 9]. Such approaches essentially extend the property graph model rather than creating a truly unified model. As a result, time-series capabilities are limited in terms of analysis and querying, and there remains a dearth of operators and algorithms that fully leverage both data types in tandem. Although some domain-specific machine learning models combine those data types [10, 11, 12, 13], a general-purpose approach is lacking.

Through HyGraph we aim to provide a unified system that handles the complexities of integrating graph and time series data, offering flexible functionalities. The core of HyGraph is a novel data model designed with equal emphasis on graph and time series data, enabling the development of hybrid operators, algorithms, and data mining techniques specifically tailored for this combined data structure. This model, detailed below, lays the foundation for a flexible approach to analyzing graph and time series data in unison.

Let $\mathcal{K}$ be the set of property keys, $\mathcal{N}$ the set of property values, $\mathcal{L}$ the set of labels and $\mathcal{T}$ the set of timestamps.

Definition 1. ***Temporal Property Graph (TPG).*** We reference the property graph model defined in [14] and extend it by adding a validity period for each element. A TPG $\mathcal{G}$ can be represented by a tuple as, $\mathcal{G} = (V_{pg}, E_{pg}, \rho, \phi, \lambda, \eta)$, where:

Published in the Proceedings of the Workshops of the EDBT/ICDT 2025 Joint Conference (March 25-28, 2025), Barcelona, Spain
*Corresponding author.
ammar@informatik.uni-leipzig.de (M. Ammar);
shubhangi.agarwal@liris.cnrs.fr (S. Agarwal);
angela.bonifati@univ-lyon1.fr (A. Bonifati);
rahm@informatik.uni-leipzig.de (E. Rahm)
0009-0005-8959-3643 (M. Ammar); 0009-0004-4405-4833 (S. Agarwal); 0000-0002-9582-869X (A. Bonifati); 0000-0002-2665-1114 (E. Rahm)

- V_{pg} and E_{pg}: Sets of vertices and edges respectively.
- ρ: $E_{pg} \rightarrow V_{pg} \times V_{pg}$ maps an edge to its source and target vertices.
- $\phi : (V_{pg} \cup E_{pg}) \times \mathcal{K} \rightarrow \mathcal{N}$ is a property function mapping each graph element and property key $k \in \mathcal{K}$ to a property value in $\mathcal{N}$.
- $\lambda : (V_{pg} \cup E_{pg}) \rightarrow \mathcal{L}$ associates each graph element with a unique label from the set of labels $\mathcal{L}$.
- $\eta : (V_{pg} \cup E_{pg}) \rightarrow \mathcal{T} \times \mathcal{T}$ retrieves the start and the end timestamps between which the graph element is valid. Let $\{t_{\text{start}}, t_{\text{end}}\} \in \mathcal{T}$ represent the two timestamps, then $t_{\text{start}} \prec t_{\text{end}}$, where the symbol $\prec$ specifies ordering, (t_{end} is initialized to $max(\mathcal{T})$).

Definition 2. ***Time series.*** A time series ts (univariate or multivariate) is an ordered set of tuples $ts = \{(t_1, y_1), (t_2, y_2), \ldots, (t_n, y_n) | n \in \mathbb{N}\}$, with timestamp $t_i \in \mathcal{T}$, such that $t_i \prec t_j$ if $i < j$, and y_i represents a tuple of real values $y_i = (val_{i_1}, val_{i_2}, \ldots, val_{i_k})$.

Definition 3. ***Dynamic Subgraph.*** Let $s \in S$ be a subgraph where S represents a set of subgraphs. The function $\psi : S \times \mathcal{T} \rightarrow \mathcal{P}(V_{pg}) \times \mathcal{P}(E_{pg})$ maps a subgraph at a time $t \in \mathcal{T}$ to a set of constituent vertices and edges from V_{pg} and E_{pg}, respectively, while $\mathcal{P}(\cdot)$ denotes the power set.

Further, two subgraphs may overlap at any point in time, $t \in \mathcal{T}$. The overlap between two subgraphs $\{s_1, s_2\} \in S$ can then be captured as the set of vertices and edges common to both the subgraphs at t, i.e., $\alpha(s_1, s_2, t) = \{(\pi_v(s_1, t) \cap \pi_v(s_2, t)), (\pi_e(s_1, t) \cap \pi_e(s_2, t))\}$. Here, $\pi_v : S \times \mathcal{T} \rightarrow \mathcal{P}(V_{pg})$ and $\pi_e : S \times \mathcal{T} \rightarrow \mathcal{P}(E_{pg})$ are projection functions that retrieve the set of constituent vertices and edges, respectively, for a subgraph at any given time.

We extend the property-graph model to incorporate time-series data, such that any vertex, edge, or subgraph can hold time-series properties. Formally, we expand the scope of sets of property keys and values to include both *static* and *dynamic*, thus embedding time series as a natural property.

Definition 4. ***Property.*** The property of a graph element is represented by a key-value pair, where the key and value belong respectively to the set of keys $\mathcal{K}$ and values $\mathcal{N}$, respectively. The map function $\phi : (V_{pg} \cup E_{pg} \cup S) \times \mathcal{K} \rightarrow \mathcal{N}$ maps a vertex, edge or a subgraph and a property key to a property value in $\mathcal{N}$, where $\mathcal{N} = \{\mathcal{N}_\Sigma \cup \mathcal{N}_{\text{TS}} \mid \mathcal{N}_\Sigma \cap \mathcal{N}_{\text{TS}} = \varnothing\}$. The set $\mathcal{N}_\Sigma$ is the set of all possible static property values and the set $\mathcal{N}_{\text{TS}}$ contains the dynamic property values, i.e., time series. Dynamic properties are further classified into two categories:

- *Regular Properties.* These store external data associated with the object, representing attributes that evolve based on external updates or observations.
- *Meta-Properties.* These store internal data derived from the graph itself, such as the evolution of graph metrics (e.g., node degree, centrality measures) or aggregated properties over edges (e.g., traffic volume between nodes). These meta-properties provide insights into the graph's internal structure and dynamic behavior.

Now that we have established the fundamental definitions, we formally introduce the HyGraph model, detailing its structural components and integration of property graphs and time series data models.

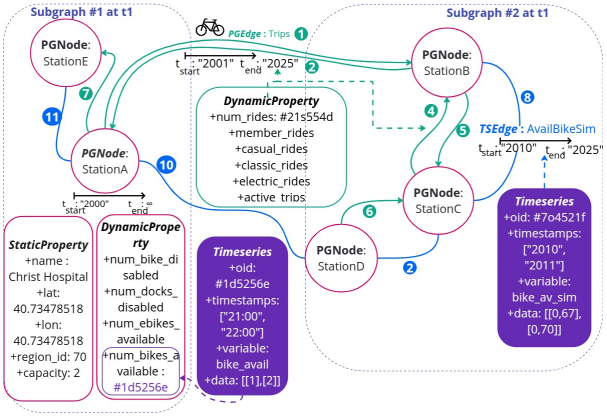

Figure 1: Example HyGraph snapshot of bike-sharing data

Definition 5. ***HyGraph Model.*** The HyGraph model is denoted by a tuple,

$$HG = (V, E, S, TS, \rho, \phi, \delta, \psi, \lambda, \eta)$$

where, V is the set of vertices, E the set of edges, S the set of logical subgraphs and TS the set of time series.

- V: A union set of property graph vertices (V_{pg}) and time series vertices (V_{ts}), i.e., $V = V_{pg} \cup V_{ts}$.
- E: Similar to V, it is defined as a union set of property graph and time series edges, i.e., $E = E_{pg} \cup E_{ts}$.
- The function $\delta : (V_{ts} \cup E_{ts}) \rightarrow TS$, maps a time-series vertex and edge to a multi-variate time series in TS.

All the mapping functions are adapted to include both property graph and time series graph objects.

- $\rho : E \rightarrow V \times V$ maps edges to source and target vertices.
- $\phi : (V \cup E \cup S) \times \mathcal{K} \rightarrow \mathcal{N}$. The map function is modified to include a subgraph, which is treated as a logical graph object and can have associated properties.
- The subgraph mapping function is adapted to allow a subgraph to have edges and vertices of both types, property graph, and time series, as, $\psi : S \times T \rightarrow \mathcal{P}(V) \times \mathcal{P}(E)$.
- The label function $\lambda : (V \cup E \cup S) \rightarrow \mathcal{P}(\mathcal{L})$ associates an entity with labels from the set of labels $\mathcal{L}$.
- Finally, the function $\eta : (V \cup E \cup S) \rightarrow \mathcal{T} \times \mathcal{T}$ retrieves the start and the end timestamps between which a graph element is valid.

This new extension ensures that time series are treated as structured entities that can be queried, connected to other time series, and analyzed within a TPG framework.

Figure 1 shows an example *HyGraph* created from a snapshot of a bike-sharing system (NYC "CitiBike" [15]). In this representation, stations are modeled as property graph vertices (V_{pg}), while trips between stations are represented as property graph edges (E_{pg}). The edges, shown in green in Figure 1, encode the trips connecting two station nodes. The *AvailBikeSim* edge set is derived to represent the similarity in bike availability patterns between station nodes. An *AvailBikeSim* edge is generated when the computed time

series similarity between the corresponding dynamic properties ($num\_bikes\_available$ and $num\_ebikes\_available$) of two stations meets or exceeds a predefined threshold. The stored values represents the evolution of the similarity score (e.g., 0.67, 0.79), computed using a time series similarity method [16] (e.g., Pearson correlation). The edges of this edge set are modeled as time series edges (E_{ts}) and are depicted in blue color in Figure 1. Each *Station* node has:

1. An id, e.g., $StationA$;
2. A validity interval, e.g., $\eta(StationA) = \langle 2000, \infty \rangle$;
3. Static properties, e.g., $\phi(StationA, name)$ = Christ Hospital, $\phi(StationA, capacity) = 2$ and;
4. Dynamic properties (time-series), the time series properties are represented as an object with a list of timestamps and associated data values for the variable at each timestamp. For instance, for $StationA$ the dynamic attribute $num\_bikes\_available$ has a variable $timestamps = [\text{“}21:00\text{”}, \text{“}22:00\text{”}]$ which stores a list of timestamps, and the associated $variable$ name $bike\_avail$ holds $data = [[1], [2]]$. Note that this is a regular dynamic property as its value change due to external factors, like a trip being undertaken.

Each edge in *Trips* consists of:

1. A label, e.g., ②;
2. A validity interval, e.g., $\eta(②) = \langle 2001, 2025 \rangle$;
3. Dynamic properties (*regular*), e.g., $member\_rides$, $casual\_rides$, etc.

Each time series edge *AvailBikeSim* consists of:

1. A label, e.g., ⑧;
2. A validity interval, e.g., $\eta(⑧) = \langle 2010, 2025 \rangle$;
3. Dynamic attribute represented as an object with a list of timestamps and associated data values for the variable at each timestamp. For instance, edge ⑧ represents similarity score evolution between *StationB* and *StationC* through a list of *timestamps* [“2010”, “2011”], and the associated $variable$ name is $bike\_av\_sim$, which holds *data* [[0.67], [0.70]].

Figure 1 also shows two non-overlapping subgraphs, *Subgraph #1* and *Subgraph #2*, depicting partial views of the network at time $t1$. *Subgraph #1* includes *StationA* and *StationE*, and connecting edges, and *Subgraph #2* features $StationB$, $StationC$ and $StationD$, and connecting edges.

Since the subgraphs can evolve with time and are dynamic, Figure 1 only depicts a snapshot at t_1. The logical subgraphs aggregate vertices that share the same *AvailBikeSim* patterns. For instance, *StationB* will continue to be a part of *Subgraph #2* till it has a high similarity score, but as soon as the score decreases it can be placed out or moved to another subgraph.

3. HyGraph Architecture

We aim to seamlessly integrate time series and graph components in a single system that allows combined querying and transformations over these components. At the moment, theHyGraph system is being developed as a Python package

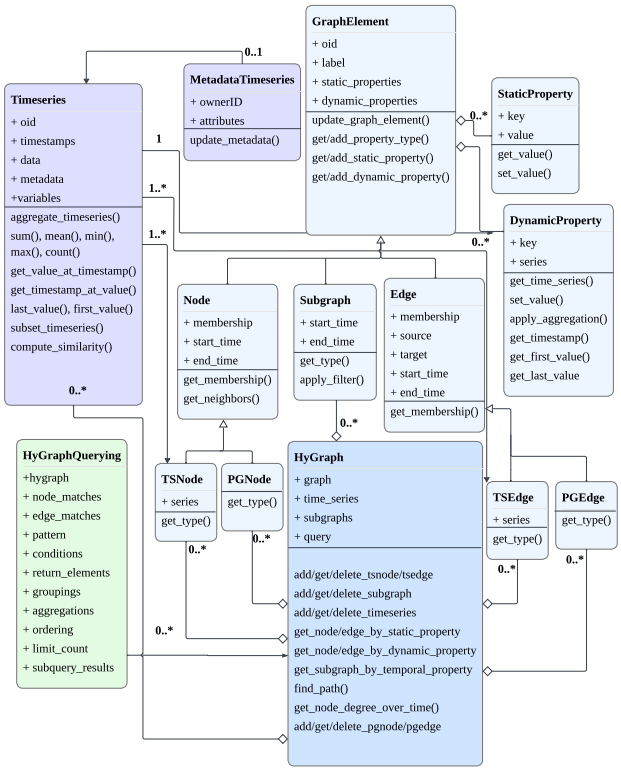

Figure 2: HyGraph UML conceptual diagram

with everything handled in memory, using NetworkX [17] and Xarray [18] for graph and time series in-memory storage. While scalability to larger storage engines is desirable, the current focus is on proof-of-concept, emphasizing a uniform data storage and querying model. We discuss the conceptual architecture of the HyGraph model through a UML diagram, followed by its system architecture with some core functionalities.

3.1. UML Architecture

In Figure 2, a UML diagram describes the HyGraph system representation in a conceptual aspect. It illustrates the main classes and their relationships. We capture HyGraph as a set of interrelated classes reflecting property-graph and time-series functionalities. At the root, an abstract class *GraphElement* defines fundamental attributes (such as *id*, and *label*). Three main classes inherit from *GraphElement*: *Node*, *Edge*, and *Subgraph*. Each of the classes plays core structural roles. Within this architecture, the class *Node* is inherited by classes *PGNode* (representing standard property-graph nodes) and *TSNode* (representing time-series nodes), while the class *Edge* is similarly inherited by classes *PGEdge* and *TSEdge*.

The *Timeseries* class defines a multivariate time series by maintaining five attributes, *id*, *data* to capture the value of multi-variate time series at different points in time, *variables* holds information about each dimension of the multivariate time series, *timestamps* to hold an ordered list of timestamps for all recorded entries in *data* and the fifth attribute is the *metadata*. The *metadata* attribute, which is an instance of a separate class *MetadataTimeseries*, is optional and facilitates additional descriptive attributes.

A top-level *HyGraph* class aggregates all these components, ensuring that property-graph elements (*PGNode*,

PGEdge) and time-series elements (*TSNode*, *TSEdge*) coexist in one coherent framework. *Node* and *Edge* and *Subgraph* have two attributes *start_time* and *end_time* as timestamps, to represent their time validity. *TSNode* and *TSEdge* classes hold in addition an instance of the *Timeseries* class as attributes to store the time series. The classes *StaticProperty* and *DynamicProperty* represent the two types of properties. Specifically, an object of class *GraphElement* can hold none or multiple property instances. The variables *static_properties* and *dynamic_properties* in class *GraphElement*, respectively represent instances of classes *StaticProperty* and *DynamicProperty* in Figure 2. The class *StaticProperty* captures properties with static values (represented by $\mathcal{N}_\Sigma$). It simply stores the *key* and its corresponding *value* as attributes. While the class *DynamicProperty* corresponds to a property whose values evolve (represented by $\mathcal{N}_{\text{TS}}$). Thus, in addition to the *key*, it references an ID of the related time series instance. The *GraphElement* class manages these properties, exposing the logic to read, insert, delete, or modify them.

The dynamic nature of subgraphs is captured via the attribute *membership* in *Node* and *Edge*, implemented as an instance of the *Timeseries* class. Concretely, an object of either class can accumulate multiple membership updates over time - one per subgraph change.As a result, every inclusion or exclusion is appended to *membership*, effectively reflecting how the subgraph's composition evolves. By tying membership changes to a time-series structure, we maintain a complete history of when a node or edge was valid in each subgraph. This allows subgraphs to evolve as entities join, leave, and transform with updates to the *HyGraph* object.

The *HyGraphQuerying* class provides a hybrid pattern-matching mechanism, allowing users to define queries that simultaneously reference graph and time-series patterns. The class supports key concepts like node- or edge-based matching, groupings, and aggregations, reminiscent of Cypher-style clauses.

3.2. System Architecture and Functionalities

Figure 3 shows a high-level architecture of the HyGraph system, illustrating interactions between different layers. It stores and processes both graph and time-series elements in memory.

We leverage two specialized processors: NetworkX as the graph processor, for structural operations (e.g., shortest paths, subgraph extraction, etc.) and; Xarray as the time-series processor, suitable for multivariate data storage and computations. We extend the pre-defined object models provided by these libraries to implement our custom-defined classes (described in Section 3.1). An object-mapping strategy then bridges these customized classes with the underlying library objects, enabling both data persistence and data retrieval. High-level algorithms such as *hybrid pattern matching* or *graph similarity*, are developed on top of this system, to enable analyses of the hybrid data. In the implementation of the HyGraph system, we adopt a modular architecture with six principal modules, orchestrated by the main *HyGraph* module. The *GraphOperator* module handles all graph-centric logic, in addition to integrating graph and time series operations, ensuring that HyGraph supports standalone property graph or time series features. This module implements the classes *GraphElement*, *Node*, *Edge*, *Subgraph*, *StaticProperty*, and *DynamicProperty*. In parallel, the *TimeseriesOperator* module manages time series

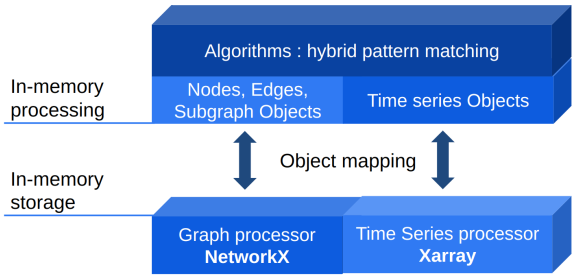

Figure 3: HyGraph System Architecture

functionalities, implementing classes like *Timeseries* and *MetadataTimeseries* for storing and manipulating temporal data. To facilitate the import and export of data, the system is equipped with the *HyGraphFileLoader* module, which aids in streamlining ETL (Extract–Transform–Load) tasks. In our system, we also define a module *GraphConstruct*. This module facilitates (re-)construction of a graph purely from a correlation between time series. This enables transformations that spawn graph structures based on temporal similarity. The module *HyGraphQuery* is defined to combine all types of querying within HyGraph. At the moment it already includes the *HybridPatternMatching* class, and will also include as a future work other classes like *SubgraphMatching* to enable searching for patterns corresponding to a whole subgraph. All these modules interoperate under the central *HyGraph* module, which coordinates their interactions and maintains the global system state.

The HyGraph system's functions are intuitively grouped into three principal interfaces, as shown in Table 1. The interface *ModelToHyGraph* gathers data from an external model (graph-only or time-series–only) and ingests it into *HyGraph*. Here, *GraphOperator* handles graph injection, adding nodes, edges, and their properties. In parallel, *TimeseriesOperator* manages time-series injection, creating or updating dynamic properties and elements. This interface also includes the Graph similarity generation function as part of *GraphConstruct* module, which will be explained later in Section 4.2. Note that *Model* refers to graph or time-series data models.*GraphConstruct* can also generate another The second interface, *HyGraphToHyGraph*, comprises the core operations and algorithms that transform one *HyGraph* into another, such as hybrid pattern matching, dynamic subgraph creation, and HyGraph clustering. *HyGraph* instance from the existing one. Finally, the third interface, *HyGraphToModel* focuses on extracting or exporting data back into an external format or distinct model. *GraphOperator* can provide standalone graph operators such as get the neighbors, and shortest path, while *TimeseriesOperator* handles isolated time-series operations such as correlation and feature extraction.

4. HyGraph Transformation Operators

Data transformations may convert one representation into another or produce a new instance in the same representation (augmented, summarized, or updated). These transformations can be static (one-off) or dynamic (continuously adapting as data changes). Within HyGraph, we distinguish two primary types of transformations. The first transformation type is from time series to graph. It involves analyzing correlations and other temporal relationships among

Table 1
HyGraph modules and associated functionalities provided for different interfaces

Modules	Interfaces		
	ModelToHyGraph	**HyGraphToHyGraph**	**HyGraphToModel**
GraphOperator	Graph injection	Subgraph creation, Clustering	Standalone graph operators
TimeseriesOperator	Time series injection	–	Standalone Time series operators
HyGraphQuery	–	Hybrid pattern matching	Data extraction and retrieval
GraphConstruct	Graph similarity generation	Graph similarity generation	-

time series to generate new graph entities (nodes, edges, or subgraphs) that reflect these relationships. The second transformation type transforms a graph into a time series. By examining evolving graph metrics (like node in-degree or edge traffic), construction of new time series that capture these structural changes over time. Moreover, HyGraph supports the continuous execution of these transformations. If the graph is updated, the transformed time series can be updated simultaneously, and vice versa.

In the following subsections, we illustrate two transformations: (i) a time-series–based similarity graph operator (Section 4.1), and (ii) an extraction operator to extract time series from evolving graph metrics (Section 4.2). These examples demonstrate the support for flexible and bidirectional transformations that unify structural and temporal data in a coherent ecosystem of HyGraph.

4.1. Time series based Graph Similarity

Several existing methods already construct graphs from time-series data for tasks like clustering or anomaly detection [19, 20, 21, 22]. They typically compute pairwise similarities or distances among time series and generate a static graph whose edges represent these similarities. In contrast, HyGraph provides a similar time series similarity-to-graph mechanism and also integrates the newly created HyGraph within the unified hybrid system. This implies that the resulting HyGraph can also maintain static and dynamic properties on edges, and be used for further processing by hybrid operators, like pattern matching.

We define our time series-based graph similarity operator formally as a function $GraphSim(TS, methods, \theta) \rightarrow H'$ where TS is a set of time series, $methods$ is a set of similarity strategies (*correlation*, *shape similarity*, etc.) and $\theta \in [0, 1]$ is a similarity threshold. The output H' is a new HyGraph instance generated by analyzing time series nodes. Specifically, let $\{v_1, v_n\} \subseteq V_{\text{ts}}$ represent time series nodes, then for each edge, represented as a vertex pair (v_i, v_j), we compute the similarity score of their time series as $Sim_{ts}(v_i, v_j)$. If $Sim_{ts}(v_i, v_j) \geq \theta$, an edge is created. The similarity score is stored as the static or dynamic edge property. If the user only requests a single, fixed value, a *PGEdge* is created with a static property. However, if the evolution of similarity over time is of interest, it is more strategic to store it as a time series in an instance of *TSEdge*.

The objective of the operator is to either create a *HyGraph* from scratch when only time-series data is provided, or to further analyze time series in the existing *HyGraph* instance by applying graph operators to time series. In HyGraph terminology, this is a *ModelToHyGraph* transformation, where one or more time series (either ingested from an external source or extracted from the current *HyGraph*) are analyzed to produce a *HyGraph* reflecting their interrelationships.

4.2. From Graph Topology to Time series

Prior approaches have examined how graph metrics evolve, by either implementing algorithms that always catch new changes in the graph structure and update the results of the graph operator [23] or by creating time series to analyze patterns [24, 25]. However, most solutions stop after generating standalone time series data and do not further link it with the graph. In HyGraph, we can generate time series by analyzing the evolution of graph topology and optionally, embed them back into the HyGraph, either as dynamic property of existing graph elements or as dedicated element (as instances of *TSNode* or *TSEdge*). This allows transformation of the HyGraph through augmentation of the derived data and enables further transformation operations. We define the extraction operator as:

$$ExtractTS(H, \mathcal{F}, metric, \tau, freq) \rightarrow \{ts_1, ..., ts_m\} \cup H'$$

where,

- H is a *HyGraph* instance,
- $\mathcal{F}$ is a filter (or set of filters) that specifies which vertices/edges or subgraphs to evaluate (e.g., node filter based on labels),
- $metric$ specifies the graph property over which the time series is to be generated, e.g., degree centrality, clustering coefficient, etc.
- $\tau = [t_{\text{start}}, t_{\text{end}})$ specifies a time range, $\{t_{\text{start}}, t_{\text{end}}\} \in \mathcal{T}$,
- $freq$ indicates sampling frequency (e.g., daily, weekly).

We enumerate discrete time steps at $freq$ in the time range τ, as $\{t_1, t_2, \ldots\}$. At each time step t_i, we take a snapshot of the *HyGraph* instance. The value for $metric$ is then computed for each snapshot and assembled into a time series. The time series thus generated reflects the evolution of $metric$ across the selected nodes/edges over discrete time intervals. The time series can be processed for further analysis, or injected back into the *HyGraph* instance as dynamic properties of the graph elements, to produce an updated instance, or can simply be returned as a set of time series.

5. Use case: Micro-mobility

Micro-mobility has emerged as a cornerstone of sustainable urban transportation. Yet, one of its persistent operational hurdles is rebalancing, ensuring that vehicles such as bicycles or e-bikes are appropriately distributed across docking stations to meet fluctuating demand. Studies focusing on bike-sharing systems emphasize that neglected rebalancing can lead to chronic station shortages or overflows, hindering overall service reliability and increasing user dissatisfaction [26, 27, 28].

To address the rebalancing challenge, we propose a multi-step pipeline that leverages HyGraph's transformation operators, *GraphSim* and *ExtractTS* (described in Section 4). Our core objective is to determine, for each station, which other station(s) serve as ideal rebalancing partners, i.e., whenever one station experiences a surplus, the other experiences a deficit, while simultaneously accounting for neighbor connectivity and distance. We base our analysis on the dataset provided by [15]. It unifies graph and time-series data by representing a bike station as a vertex with a static property representing the parking capacity of the station and dynamic properties like the number of available bikes; while each edge represents trips between two stations, a time-series property tracking daily active trips, member rides vs. casual rides, and total trips.

Using the ExtractTS operator, we first extract two dynamic properties for each station node:

1. For a station v at time t, the imbalance is defined as the difference between the number of trips that end at station v (i.e. bikes arriving) until time t, and the number of rides starting from station v (i.e. bikes departing) until time t. The imbalance value is for a station is captured at different timestamps and is stored as a time series property, $imbalance\_ts$.

2. For a station v, we also compute its connectivity score to quantify how strongly it is connected to its neighbors. The connectivity score for v is defined as the ratio of weighted sum of edges and the degree of v at any time t. Similar to imbalance of a station, the connectivity score is also stored as a dynamic property, $connectivity\_ts$, of the vertex.

In the next step, a similarity graph is constructed using the module *GraphConstruct*, where nodes represent stations and edges represent the similarity of their time series property $imbalance\_ts$. To capture the complementary behavior of two stations, i.e., pairing a surplus station with a deficit station, we compute a negative correlation between $imbalance(v, t)$ for station v and $imbalance(u, t)$ for station u. The function will augment the *HyGraph* instance with new *TSNode* objects, created to represent the $imbalance\_ts$ of each station and new *PGEdge* objects representing the similarity between the newly created *TSNode* objects. The negative correlation between the imbalance time series of two stations u and v, $neg_{\text{imb}}(u, v)$ quantifies how complementary the two stations are.

After building the similarity graph, for every edge connecting stations u and v, we compute a composite score that will represent the weight of the edge. This weight is a combination of the following: a similarity score based on a distance decay function [29], the distance between the two stations, the negative correlation score $neg_{\text{imb}}(u, v)$ and the average value of $connectivity\_ts$ between the two stations. This composite score reflects both the temporal complementarity of imbalance and the practical factors of connectivity and distance. Once the similarity graph is fully augmented, we apply a maximum weighted matching algorithm [30] to select a set of non-overlapping edges and return the set of station pairs (u, v) that maximize the total composite score.

For each matched pair, the average imbalance difference is computed to suggest the direction and the number of bikes that should be transferred from one station to another.

6. Future Research

HyGraph represents an initial step toward integrating property graphs with time-series, addressing key challenges in maintaining and querying dynamic data. By unifying these two paradigms, HyGraph enables seamless temporal graph transformations, but its implementation also presents several complexities.

However, one major challenge lies in efficiently updating and querying time-series data associated with graph nodes and edges. Indexing strategies in traditional graph databases are not inherently designed to accommodate time-series data efficiently, leading to potential scalability bottlenecks. A new system that integrates indexing techniques tailored for both graph structures and time-series storage, would ensure efficient querying and seamless data evolution. Maintaining indexing structures that accommodate both topological changes in the graph and temporal variations in time-series data requires novel optimization techniques.

Additionally, the lack of a standardized query language for seamlessly integrating time-series operations with graph traversal necessitates the design of new operators and query execution strategies. Existing graph query languages do not natively support analytic operations commonly found in time-series databases, such as temporal aggregations, windowed computations, and similarity searches based on sequence patterns or shape-based matching. Future research could explore the development of a unified query language that incorporates time-aware traversal semantics and transformation operators to enable efficient interaction between graph topology and temporal dynamics.

The fast-evolving nature of time-series data necessitates low-latency updates and retrieval, making it essential to scale HyGraph for real-time applications. Addressing this challenge requires investigating efficient data streaming architectures, like designing caching mechanisms for frequently queried data and hybrid storage layouts optimized for high-throughput ingestion and query concurrency.

7. Conclusion

This paper introduced the UML and sytem architecture of HyGraph [7], illustrating a unified approach for integrating property graphs and time-series data. We introduced two novel transformation operators: (i) a time-series-based graph operator, which derives graphs based on correlations among time series, and (ii) a graph-based time-series operator, which extracts time-series representations from evolving graph metrics.

Our micro-mobility case study further demonstrated the practical applicability of HyGraph and the transformation operators for augmented analysis in real-world settings. By establishing a foundation for hybrid graph-time-series analytics, HyGraph paves the way for plethora of research opportunities in graph data management, temporal reasoning, and dynamic query processing.

Despite its advantages, several challenges remain and future research should explore scalable indexing and query optimization techniques for hybrid queries.

References

[1] S. Bhandari, N. Bergmann, R. Jurdak, B. Kusy, Time series data analysis of wireless sensor network mea-

surements of temperature, Sensors 17 (2017) 1221.
[2] R. Krishnamurthi, A. Kumar, D. Gopinathan, A. Nayyar, B. Qureshi, An overview of iot sensor data processing, fusion, and analysis techniques, Sensors 20 (2020) 6076.
[3] H. Zhang, J. Chen, W. Li, X. Song, R. Shibasaki, Mobile phone gps data in urban ride-sharing: An assessment method for emission reduction potential, Applied Energy 269 (2020) 115038.
[4] L. Belkessa, M. Ameli, M. Ramezani, M. Zargayouna, Multi-channel spatio-temporal graph convolutional networks for accurate micromobility demand prediction integrating public transport data, in: Proceedings of the 2nd ACM SIGSPATIAL Workshop on Sustainable Urban Mobility, 2024, pp. 5–13.
[5] A. Bader, O. Kopp, M. Falkenthal, Survey and comparison of open source time series databases, in: Datenbanksysteme für Business, Technologie und Web (BTW 2017) - Workshopband, Gesellschaft für Informatik e.V., Bonn, 2017, pp. 249–268.
[6] K. Mishra, S. Basu, U. Maulik, Graft: A graph based time series data mining framework, Eng. Appl. Artif. Intell. 110 (2022).
[7] M. Ammar, C. Rost, R. Tommasini, S. Agarwal, A. Bonifati, P. Selmer, E. Kharmlamov, E. Rahm, Towards hybrid graphs: Unifying property graphs and time series, 28th International Conference on Extending Database Technology (2025).
[8] E. Bollen, R. Hendrix, B. Kuijpers, Managing data of sensor-equipped transportation networks using graph databases, Geoscientific Instrumentation, Methods and Data Systems Discussions 2024 (2024) 1–30. URL: https://gi.copernicus.org/preprints/gi-2024-3/. doi:10.5194/gi-2024-3.
[9] B. Steer, F. Cuadrado, R. Clegg, Raphtory: Streaming analysis of distributed temporal graphs, Future Generation Computer Systems 102 (2020) 453–464.
[10] J. Chen, X. Wang, X. Xu, Gc-lstm: Graph convolution embedded lstm for dynamic network link prediction, Applied Intelligence (2022) 1–16.
[11] S. Bloemheuvel, J. van den Hoogen, D. Jozinović, A. Michelini, M. Atzmueller, Graph neural networks for multivariate time series regression with application to seismic data, International Journal of Data Science and Analytics 16 (2023) 317–332.
[12] S. Gocheva-Ilieva, H. Kulina, A. Yordanova, Stacking machine learning models using factor analysis to predict the output laser power, in: 2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), IEEE, 2022.
[13] Z. Wang, H. Ren, R. Lu, L. Huang, Stacking based lightgbm-catboost-randomforest algorithm and its application in big data modeling, in: 2022 4th International Conference on Data-driven Optimization of Complex Systems (DOCS), IEEE, 2022, pp. 1–6.
[14] R. Angles, The property graph database model, in: D. Olteanu, B. Poblete (Eds.), Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, May 21-25, 2018, volume 2100 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2018.
[15] Lyft Bikes & Scooters, C. Urbainsky, New york city bike sharing network: Time-series enhanced nodes and edges dataset, 2024. URL: https://doi.org/10.5281/zenodo.13846868. doi:10.5281/zenodo.13846868, accessed: 2024-09-27.
[16] A. Kianimajd, M. G. Ruano, P. Carvalho, J. Henriques, T. Rocha, S. Paredes, A. E. Ruano, Comparison of different methods of measuring similarity in physiologic time series, IFAC-PapersOnLine 50 (2017) 11005–11010.
[17] NetworkX, Networkx: Network analysis in python, 2024. URL: https://networkx.org/.
[18] Xarray, Xarray: Dealing with multidimensional arrays in python, 2024. URL: https://docs.xarray.dev/en/stable.
[19] D. Tiano, A. Bonifati, R. Ng, Featts: Feature-based time series clustering, in: G. Li, Z. Li, S. Idreos, D. Srivastava (Eds.), SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021, ACM, 2021, pp. 2784–2788.
[20] P. Li, S. F. Boubrahimi, S. M. Hamdi, Graph-based clustering for time series data, in: 2021 IEEE International Conference on Big Data (Big Data), IEEE, 2021, pp. 4464–4467.
[21] L. N. Ferreira, L. Zhao, Time series clustering via community detection in networks, Information Sciences 326 (2016) 227–242.
[22] K. F. Eteffa, S. Ansong, C. Li, M. Sheng, Y. Zhang, C. Xing, An experimental study of time series based patient similarity with graphs, in: Web Information Systems and Applications: 17th International Conference, WISA 2020, Guangzhou, China, September 23–25, 2020, Proceedings 17, Springer, 2020.
[23] D. Eppstein, Z. Galil, G. F. Italiano, Dynamic graph algorithms, Algorithms and theory of computation handbook 1 (1999) 9–1.
[24] C. Aggarwal, K. Subbian, Evolutionary network analysis: A survey, ACM Computing Surveys (CSUR) 47 (2014) 1–36.
[25] C. Rost, K. Gomez, P. Christen, E. Rahm, Evolution of degree metrics in large temporal graphs, in: Datenbanksysteme für Business, Technologie und Web (BTW 2023), volume P-331 of *LNI*, Gesellschaft für Informatik e.V., 2023, pp. 485–507. URL: https://doi.org/10.18420/BTW2023-23. doi:10.18420/BTW2023-23.
[26] K. Wang, X. Yan, Z. Zhu, X. M. Chen, Understanding bike-sharing usage patterns of members and casual users: A case study in new york city, Travel Behaviour and Society 36 (2024) 100793.
[27] Y.-T. Hsu, L. Kang, Y.-H. Wu, User behavior of bikesharing systems under demand–supply imbalance, Transportation Research Record 2587 (2016) 117–124.
[28] F. Chiariotti, C. Pielli, A. Zanella, M. Zorzi, A dynamic approach to rebalancing bike-sharing systems, Sensors 18 (2018) 512.
[29] M. Halas, P. Klapka, Spatial influence of regional centres of slovakia: analysis based on the distance-decay function, Rendiconti Lincei 26 (2015) 169–185.
[30] B. Wu, L. Li, Solving maximum weighted matching on large graphs with deep reinforcement learning, Information Sciences 614 (2022) 400–415.