Temporal Graph Analysis using Gradoop

Christopher Rost[1] Andreas Thor[2] Erhard Rahm[3]

Abstract: The temporal analysis of evolving graphs is an important requirement in many domains but hardly supported in current graph database and graph processing systems. We therefore have started with extending Gradoop for temporal graph analysis by adding time properties to vertices, edges and graphs and using them within graph operators. We outline these extensions and show their use in a bibliographic scenario to analyze temporal citation patterns.

Keywords: Temporal Graph; Temporal Graph Data Model; Graph Analysis

1 Introduction

The flexible analysis of graph data has gained significant interest in the last decade and is supported by graph database systems (e.g., Neo4j) and a growing number of distributed graph processing systems [Ju17]. While graphs typically evolve continuously, graph processing systems mostly focus on the analysis of static graphs representing the state (or snapshot) of a graph at a specific point in time. Changes like the addition of vertices and edges can occur comparatively slowly (e.g., in bibliographic networks) or at high frequency (e.g., as a stream of posts in a social network). An important requirement in many domains is to analyze the temporal dimension of graphs, e.g., to analyze the evolution of certain relationships like the citation patterns of publications or the development of co-authorships in bibliographic networks. For streaming-like changes there are specific analysis requirements, in particular to support fast, real-time reaction to certain changes such as the spread of hate messages in social networks.

In this short paper, we report on work in progress on temporal graph analysis using Gradoop [Ju16, Ju18], a distributed, open source framework for graph analytics. It supports extended property graphs as well as many declarative operators, e.g., for pattern matching and structural grouping, that can be used to realize complex workflows for graph analysis. Inspired by the temporal extensions in SQL:2011 [KM12] we extend the Gradoop graph data model by time properties for valid and transactional time. We also show how these temporal properties can be used within the operators for temporal graph analysis. Furthermore, we sketch the use of these operators for a bibliographic use case to find certain

[1] University of Leipzig, rost@informatik.uni-leipzig.de
[2] Leipzig University of Telecommunications, thor@hft-leipzig.de
[3] University of Leipzig, rahm@informatik.uni-leipzig.de

doi:10.18420/btw2019-ws-11

temporal citation patterns. After a discussion of related work we introduce the temporal extensions of Gradoop's property graphs (Sect. 3) and operators (Sect. 4). We then show the use of the operators in building blocks for common tasks in temporal graph analytics (Sect. 5) and outline the bibliographic use case in Section 6.

2 Related Work

Date et al. [DDL02] define three classes of time aspects for temporal relational databases that can be applied one-to-one to temporal graphs: *transaction time*, *valid time*, and their combination *bitemporal data*. *Transaction time* is defined as the time interval in which a fact is considered true in the database (graph). In most cases, the transaction time is maintained by the processing system itself and can be used for versioning so that graph states can be reconstructed for any point in the past. The *valid time* is defined as a time interval in which a fact is valid as defined by the context of the data [DDL02]. Valid time intervals can also be expressed by *time-stamps* if the duration can be neglected. Such graphs are also known as *transient* [KD13] or *contact sequences* [HS12] since their time-stamps reflect a chronological order of interactions (e.g., edge additions). Figure 1 shows an example of a temporal graph with valid times: The temporal affiliations of authors to institutions is represented as time intervals whereas the valid time of publications is the time-stamp when the publication was published.

There are only few graph processing systems that natively support the storage, analysis and querying of temporal graphs. *Immortalgraph* [Mi15] (earlier known as *Chronos*) provides a storage and execution engine designed for temporal graphs. It includes locality optimizations and an in-memory iterative graph computation based on series of graph snapshots. Snapshots are divided into groups to provide temporal graph mining approaches. *Kineograph* [Ch12] is a distributed platform for incoming stream data to construct a continuously changing graph. It is also based on in-memory graph snapshots which are evaluated by conventional mining approaches of static graphs (e.g., community detection). The snapshot approach is used to distribute the graph on different systems. *GraphStream* [Pi08] is an open-source Java library focusing on the dynamics aspects of a graph. It provides a flexible way to build user-defined analyses upon a dynamic graph structure based on a stream of graph events. None of these systems is based on a property graph model to hold detailed contextual information of vertices and edges besides the structural information. Then et al. [Th17] developed an automatic algorithm transformation to avoid multiple executions of graph analytics algorithms on snapshots of a temporal graph to reduce their runtimes.

A fairly new temporal graph analytics library is *Tink* [Li18] that focuses on several temporal path problems and offers the calculation of measures like temporal betweenness and closeness. Similar to Gradoop, Tink is build on Apache Flink and employs the Property Graph Model. Temporal information is represented by time intervals at the edges. In contrast to Tink, Gradoop supports not only graphs but also logical graphs and graph collections.

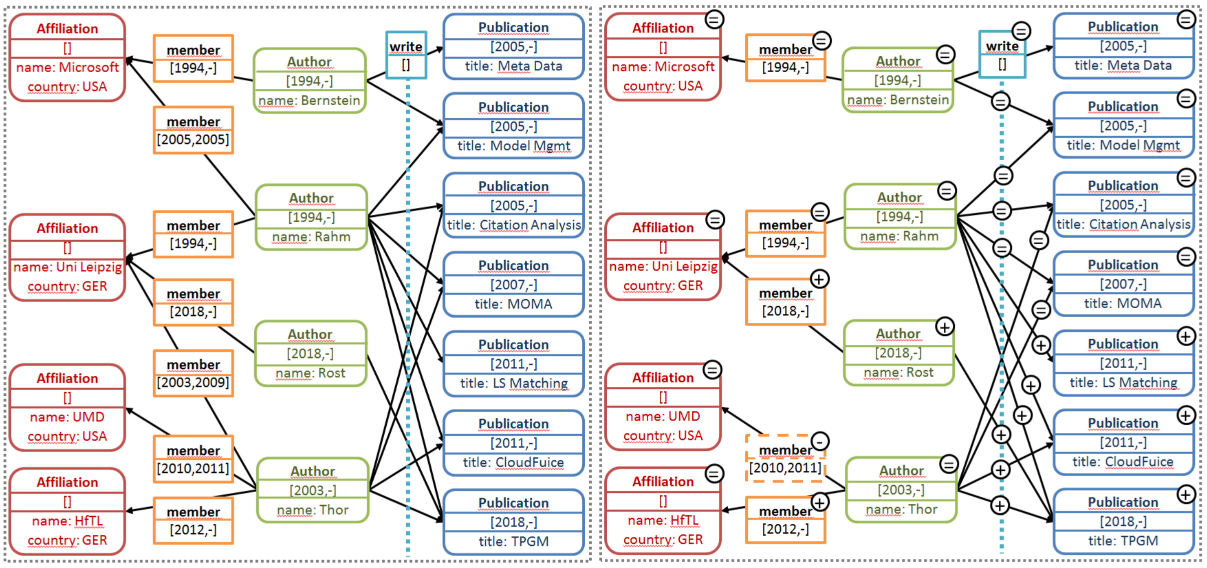

Fig. 1: TPGM example of a bibliographic graph (*left*) and the resulting annotated graph of the *Difference* operator of Sect. 4 (*right*). Vertices and edges are specified by label, valid-time (with format [*val-from,val-to*]) and properties.

Furthermore, our extension of the property graph model allows the definition of both time-stamps and intervals on both vertices and edges.

3 Temporal Property Graph Model in Gradoop

We introduce the *Temporal Property Graph Model* (TPGM) as a simple but powerful extension of the *Extended Property Graph Model* (EPGM) [Ju16] to support combinable analytical operators on directed graphs that evolve over time in Gradoop. In the EPGM, a single property graph is referred to as *logical graph*, which in turn can be part of a *graph collection*. Vertices and edges refer to one or more logical graphs and are accordingly part of them. Logical graphs, vertices, and edges consist of a unique identifier, a type label (e.g., *User* or *worksAt*), and a (possibly empty) set of properties represented as key-value pairs.

TPGM extends EPGM by adding four additional time attributes as obligatory to the schema of vertices, edges, and logical graphs: *tx-from*, *tx-to* and *val-from* and *val-to*. The first two represent the transaction time (prefix *tx*), the last two define the valid time (prefix *val*) by holding the beginning and end of the elements validity. This approach offers a flexible representation of temporal graphs with bitemporal time semantics where the valid time can be empty, a time-stamp or a time interval by setting either none, only the *val-from* or both *val-from* and *val-to* attributes. Since time attributes can be empty, also edge-centric scenarios where a graph only has time information at its edges can be modeled. Empty time attributes are interpreted as *NULL* values (e.g., in predicate functions) analogous to SQL. Fig. 1 (*left*) shows a temporal graph from the bibliographic domain modeled in TPGM. The graph represents the relationship between authors, affiliations and publications.

There are vertices and edges without a temporal specification (e.g., *Affiliation*), with a time-stamp (e.g., *Publication*) and a time interval (*member*) as valid-time. TPGM does not specify the data type of the time attributes and leaves it up to the implementation (e.g., Unix time-stamp or formatted date/time string). By holding a whole graph with both rollback and historical information, this model offers a flexible retrieval of arbitrary graph snapshots and dissociates itself from widespread snapshot approaches.

Valid times are typically embedded within the context of the data before they enter GRADOOP. The respective timestamps can be extracted while loading the elements as graph or collection into the system. The transaction times are maintained by the GRADOOP system automatically. Vertices and edges can be added to (or deleted from) a logical graph as well as a whole logical graph can be added to (or deleted from) a graph collection. For newly added graph elements the value of *tx-from* is set to the current system time (i.e., import time) and *tx-to* to infinity. If a graph element is deleted, the value of *tx-to* is set to the current system time. The deletion of a vertex automatically triggers the deletion of all corresponding edges, since they are not valid without the vertex. At the moment, the model does not support subsequent changes (updated) of an element's label, valid times or properties.

Another advantage of TPGM is its backward compatibility to the original EPGM since every EPGM operator can be applied to a TPGM graph by disregarding the temporal information of the graph elements. However, we will define appropriate operator extension for TPGM that allows the definition of temporal graph analytical workflows in the next section. Although TPGM offers bitemporal support, we limit ourselves exclusively to the valid-times for simplicity in the following sections.

4 Operators

The EPGM allows for combining multiple operators to graph analytical workflows using the domain specific language GrALa (**Gr**aph **A**nalytical **La**nguage). It already provides operator implementations for graph pattern matching, subgraph extraction, graph transformation, set operations on multiple graphs as well as property-based aggregation and selection. Some operators can be applied on logical graphs and others on graph collections [Ju16]. For simplicity we use the terms *graph* and *logical graph*, *collection* and *graph collection* interchangeably. In this work we will focus on the six graph operators that are listed in Tab. 1. They support the access and modification of the available temporal information in different ways. The following sections provide short definitions of these temporal operators with some examples. Pre-defined predicate functions that can be used by these operators are defined in Tab. 2. Furthermore, helper functions that return parts of a date or time information (e.g., year or day of week) are available. The implementation of the operators and their integration into GRADOOP is currently work in progress. Since we focus on valid-times in this section, each notation of *from* and *to* refers to the attributes *val-from* and *val-to* defined in TPGM.

Operator	Signature	Output
Transformation*	`Graph.transform(graphFunction, vertexFunction, edgeFunction)`	`Graph`
Subgraph*	`Graph.subgraph(vertexPredicateFunction, edgePredicateFunction)`	`Graph`
Snapshot	`Graph.snapshot(temporalPredicateFunction)`	`Graph`
Difference	`Graph.diff(temporalPredicateFunction, temporalPredicateFunction)`	`Graph`
Grouping*	`Graph.groupBy(vertexGroupingKeys, vertexAggregateFunction, edgeGroupingKeys, edgeAggregateFunction)`	`Graph, Collection`
Pattern Matching*	`Graph.query(patternGraph [,constructionPattern])`	`Collection`

Tab. 1: Overview of TPGM unary operators and their signature. Operators marked with * already exist in EPGM and were extended by temporal support.

Transformation. The *transform* operator defines a structure-preserving modification of graph, vertex and edge data. User-defined transformation functions can be applied to an input graph G, which results in an output graph G' [Ju16]. Within TPGM it is possible to (1) modify the temporal attributes, (2) define the time attributes from information stored in properties or (3) create properties resulting from the temporal information of the time attributes. For example, if the temporal attributes are not yet set or calculated during a workflow, this operator offers the possibility to define the valid times *from* and *to* at runtime.

Subgraph. The *subgraph* operator is used to extract a subgraph from a graph by applying predefined or user-defined predicate functions [Ju16]. Often, such a function is used to filter vertices and edges by label or the existence or value of a property (e.g., vertices with label *User* and property *age* greater than *30*). Within TPGM, the temporal information of graph elements can be used inside the predicate functions. The operator is also suitable to declare vertex-induced or edge-induced subgraphs by providing either a vertex *or* edge predicate function. Considering the graph in Fig. 1 (*left*), a subgraph with all author-affiliation memberships that last longer that 3 years can be extracted with the edge-induced operator call `graph.subgraph(null, e -> e.label = 'member' AND YEAR(e.to)-YEAR(e.from) > 3)`.

Snapshot. The *snapshot* operator allows one to retrieve a valid snapshot of the whole temporal graph either at a specific point in time or a subgraph that is valid during a given time range by providing a temporal predicate function. Besides the operator itself, several predefined predicate functions (see Tab. 2) are available. They are adopted from SQL:2011 [KM12] that supports temporal databases. In the example of Fig. 1 (*left*), `graph.snapshot(asOf(2010))` would remove the author named *Rost* and the last three publications together with their edges. Furthermore, three member edges (*Rahm-Microsoft*, *Thor-Uni Leipzig*, and *Thor-HfTL*) are removed.

Difference. The evolution of graphs over time can be represented by the difference of two graph snapshots, i.e., by a difference graph that is the union of both snapshots where each graph element is annotated as an added, deleted, or persistent element. To this end, Gradoop's structural `diff` operator consumes two graph snapshots defined by temporal predicate functions and calculates the difference graph. For example, the usage of `graph.diff(asOf(2010),asOf(2018))` at the graph in Fig. 1 (*left*) would result in the

Function	**Predicate**	
	Time-Stamp (only *from* defined)	Time-Interval (*from* and *to* defined)
asOf(x)	$from \leq x$	$from \leq x \wedge to \geq x$
fromTo(x, y)	-	$from < y \wedge to > x$
between(x, y)	-	$from \leq y \wedge to > x$
precedes(c)	$from \leq c.from$	$to \leq c.from$
succeeds(c)	$from \geq c.from$	$from \geq c.to$
overlaps(c)	-	$max(from, c.from) < min(to, c.to)$

Tab. 2: Predefined TPGM predicate functions that can be used by the operators. Variables *x* and y are timestamps, whereas *c* is a graph element. The predicates differ according to the definition of a time-stamp or a time-interval.

annotated graph at Fig. 1 (*right*). The symbols +, - and = represent added, removed and persisting elements, respectively.

Grouping. A structural grouping of vertices and edges is an important task in temporal graph analytics. Since temporal graphs can become very large, a condensation can facilitate deeper insights about structures and patterns hidden in the graph. In the current EPGM implementation of the *groupBy* operator, a grouping is based on vertex and edge grouping keys (e.g., the type label or property keys) as well as vertex and edge aggregation functions [JPR17]. For temporal grouping, TPGM provides three additional features: First, time-specific value transformation functions (e.g., year or day of week) can be applied to compute time values on the desired granularity for grouping. Second, the *groupBy* operator supports `GROUP BY CUBE` and `GROUP BY ROLLUP` similar to SQL. Third, aggregation on the temporal properties *from* and *to* of the vertices and edges can not only be specified by user-defined functions but by one of the predefined time-specific aggregation functions (e.g., *MinFrom* or *MaxFrom*). For example, the *AvgDuration* aggregate function can be used to determine the average duration of all membership edges at the graph in Fig. 1 (*left*).

Pattern Matching. Retrieving subgraphs matching a user-defined pattern graph is an important task within the graph analytics domain. In the EPGM a pattern matching operator *query* is already implemented [Ju16] using basic concepts of Neo4j Cypher[4] to define patterns, e.g., `(a)-[b]->(c)`. Predicate functions can be embedded in a pattern inside a `WHERE` clause by using variables defined in the pattern. The resulting embeddings can be modified by providing a construction pattern. In TPGM, we extend this functionality by using the temporal attributes *from* and *to* inside predicate definitions, i.e., by pre-defined (see Tab. 2) and user-defined predicate functions. For example, the pattern `(a:Author)-[m:member]->(f:Affiliation country : USA) WHERE m.asOf(2017)` describes an author being a member of an affiliation from the United States as of 2017.

[4] `https://neo4j.com/developer/cypher-query-language/`

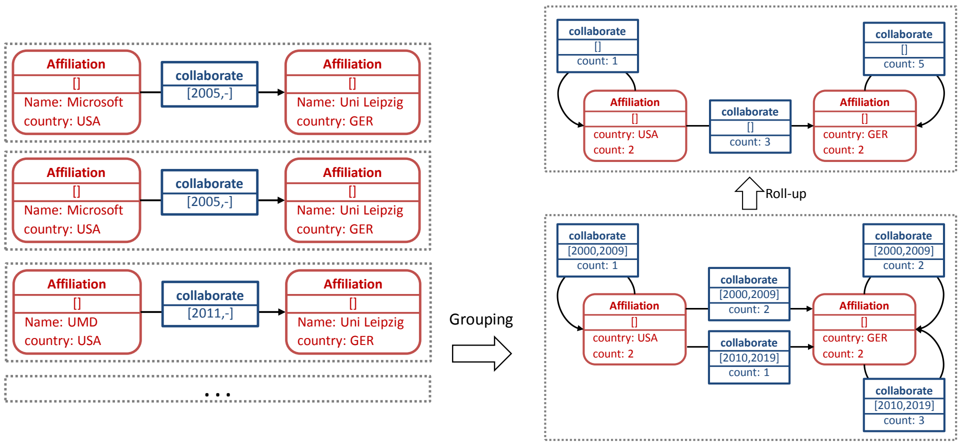

Fig. 2: Part of a graph collection containing embeddings of a temporal pattern (*left*) and the result of a time-specific grouping and aggregation (*right*) applied on these matches.

5 Temporal Graph Analytics Workflows

In this section we discuss exemplary workflows for temporal graph analytics. We illustrate how they can be supported by Gradoop and its extension to TPGM.

Snapshot generation and graph evolution: Graph systems or algorithms might focus on the analysis of static graphs representing the state (or snapshot) of a graph at a specific point in time. Gradoop therefore supports the retrieval of snapshots using the `snapshot` operator in combination with time-based predicates. To identify the difference and thus the changes between two graph snapshots, the `diff` operator can be used.

Temporal pattern matching: Searching for graph patterns using time-constraints is important for temporal analysis. In the example of Fig. 1 (*left*), a query to obtain simultaneous collaborations between affiliations from different countries must take the valid times of the *member* edges into account: `(f1:Affiliation)<-[m1:member]-(a1:Author)-[:write]->(p:Pub) (p)<-[:write]-(a2:Author)-[m2:member]->(f2:Affiliation) WHERE a1 != a2 AND m1.overlap(p) AND m2.overlap(p)`. By applying this query together with the construction pattern `(f1)-[c:collaborate{from=p.from}]->(f2)` to Gradoop's *query* operator, a collection of matching graph embeddings is generated which is exemplified in Fig. 2 (*left*). A special case of such patterns are time-respecting paths, i.e., sequences of edges with non-decreasing times that connect vertices. Time-respecting paths not only identify all vertices that can be reached from others within some observation window [HS12] (reachability) but might also define shortest paths in terms of the overall duration time of the edges.

Time-specific grouping and aggregation: The time dimension automatically introduces a hierarchy, i.e., graphs can be grouped (summarized) at multiple levels of time-granularity. For example, the graph in the bottom-right corner of Fig. 2 summarizes the collaboration between countries per decade based on co-authored publications, i.e., the *Affiliation* vertices

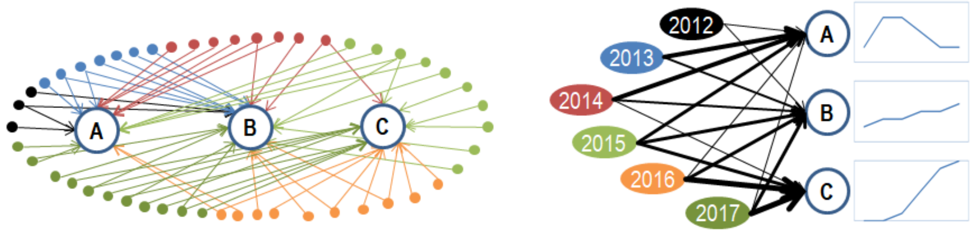

Fig. 3: Example of a citation graph (*left*) and its graph summary (*right*) for three publications *A*, *B*, and *C* that were all published in 2011. For simplicity, only citations to the three publications *A*, *B*, and *C* are illustrated, i.e., citations between the citing publications are omitted. In the graph summary, citing publications are grouped by publication year (indicated by color) and the number of citations corresponds to the line widths of the super edges. The small charts illustrate the citation dynamics: *A* follows a *common life cycle*, *B* is a *constant performer*, and *C* is a *sleeping beauty*.

are grouped by their *country* property and the *collaborate* edges are aggregated at the level of decades to reflect the temporal changes in the collaboration over time. However, this summarization can be rolled-up on the time hierarchy to have a global aggregation. To this end, GRADOOP's `groupBy` operator groups all *collaborate* edges by year with `GROUP BY ROLLUP` so that the resulting graph collection contains both graphs of Fig. 2 (*right*).

Efficient maintenance and re-use of queries and analysis results: Repeated execution of GRADOOP workflows, e.g., at certain times (e.g., once a month) or at certain events (e.g., when adding a new publication into a bibliographic network) requires an efficient update of graph transformations and graph summaries both in their structure and aggregated property values. In the above mentioned `GROUP BY ROLLUP` example, newly added *publication* vertices might only update the *collaborate* edges where the *publication* valid time falls into the valid time of the *collaborate* edge.

6 Use Case: Citation Analysis

The influence of past literature on current research is manifested by references cited in publications. Bibliographic networks therefore consist of publications (vertices) and their citations (directed edges) between them. Edges are time-stamped with the publication date of the citing publication to indicate when the citation happened. Fig. 3 shows an example with three publications *A*, *B*, and *C* (all published in 2011) that have been cited by 63 other publications. Besides the question of identifying exceptionally highly-cited publications ("top publications") it is also of interest to identify the citation dynamic of cited publications that usually follow a common *life cycle*: starting with low citations in the first or two years of publication, growing up to a maximum of citations a few years later, followed by a continuous decrease of citations several years after publication. However, other dynamics are also possible: a more or less long period of non-recognition with low citations is followed by a period with high citations after a sudden peak. Such publications are often referred to as *sleeping beauties* [vR04], i.e., they remained undetected over many years before their results, methods, ideas etc. become important for current research. In contrast, *constant performers* are characterized by citation rates which are constantly at least on the average level compared to the other publications.

In the example of Fig. 3 the citation dynamics becomes visible after grouping the citing publications by the year of publication, i.e., the summary contains six super vertices (2012-2017) where each super vertex represents all vertices (i.e., citing publications) of the corresponding year. Super edges between the super vertices and the three original nodes (A, B, and C) store the number of elements they represent, i.e., the number of citations. For example, the super edge *2017* $\rightarrow$ C represents eight edges, i.e., C has been cited by eight publications in 2017. In Fig. 3 (right) the number of citations corresponds to the line widths of the super edges. The graph summary reveals the citation dynamics of the cited publications. For example, publication C did not get any citations in 2012 and 2013, few citations in 2014 and 2015 and many in 2016 and 2017. It can therefore be classified as a *sleeping beauty*. The described use case can be expressed in GrALa as follows:

```
// subgraph including edges of last six years
sub = in.subgraph(TRUE, e => YEAR(CURRENT)-YEAR(e.from) BETWEEN 1 AND 6 )
// group citation edges by year; count number of citations
group = sub.groupBy(
   ['citingpub', (citingpub.from, t => YEAR(t))],
   (superVertex, vertices => superVertex['year'] = vertices.min(YEAR(from)),
   [:label, (from, t => YEAR(t))],
   (superEdge, edges => superEdge['count'] = edges.count()))
// call external function 'classifier' (UDF) to specify citation dynamics
group.transform(
  (gIn, gOut => gIn),
  (vIn, vOut => IF (vIn.label=='citedpub') vOut['dynamic'] = classifier(vIn)),
  (eIn, eOut => eIn))
```

7 Conclusion

We reported our work in progress on temporal graph analysis by integrating our flexible temporal property graph model TPGM that supports bitemporal time semantics into the distributed graph analytic system Gradoop. By extending the available operators, we show how the evolving nature of temporal graphs can be used to answer time-respecting analytical questions. We illustrated the use of these operators within common building blocks of analysis workflow and provided a real-world use case from the bibliographic domain to find temporal citation patterns.

Temporal graphs and their analysis is an important and promising field of research. In future work we will further extend Gradoop by temporal features such as operators and algorithms to make Gradoop a powerful and flexible system for temporal graph analysis.

Acknowledgements. This work is partially funded by Sächsische Aufbau Bank (SAB) and the European Regional Development (EFRE) under grant No. 100302179.

References

[Ch12] Cheng, R., et al.: Kineograph: taking the pulse of a fast-changing and connected world. In: Proc. EuroSys. pp. 85–98, 2012.

[DDL02] Date, C. J.; Darwen, H.; Lorentzos, N.: Temporal data & the relational model. Elsevier, 2002.

[HS12] Holme, P.; Saramäki, J.: Temporal networks. Physics Reports, 519(3):97 – 125, 2012. Temporal Networks.

[JPR17] Junghanns, M.; Petermann, A.; Rahm, E.: Distributed grouping of property graphs with GRADOOP. Proc. BTW 2017), 2017.

[Ju16] Junghanns, M.; Petermann, A.; Teichmann, N.; Gómez, K.; Rahm, E.: Analyzing extended property graphs with Apache Flink. In: Proc. SIGMOD Workshop on Network Data Analytics. 2016.

[Ju17] Junghanns, M.; Petermann, A.; Neumann, M.; Rahm, E.: Management and analysis of big graph data: current systems and open challenges. In: Handbook of Big Data Technologies, pp. 457–505. Springer, 2017.

[Ju18] Junghanns, M.; Kießling, M.; Teichmann, N.; Gómez, K.; Petermann, A.; Rahm, E.: Declarative and distributed graph analytics with GRADOOP. PVLDB, 11(12), 2018.

[KD13] Khurana, U.; Deshpande, A.: Efficient snapshot retrieval over historical graph data. In: Proc. ICDE. pp. 997–1008, 2013.

[KM12] Kulkarni, K.; Michels, J.: Temporal features in SQL: 2011. ACM Sigmod Record, 41(3):34–43, 2012.

[Li18] Ligtenberg, W.; Y.Pei; Fletcher, G.H.L.; Pechenizkiy, M.: Tink: A Temporal Graph Analytics Library for Apache Flink. In: WWW (Companion Volume). ACM, pp. 71–72, 2018.

[Mi15] Miao, Y., et. al.: Immortalgraph: A system for storage and analysis of temporal graphs. ACM Transactions on Storage, 11(3):14, 2015.

[Pi08] Pigné, Y.; Dutot, A.; Guinand, F.; Olivier, D.: GraphStream: A Tool for bridging the gap between Complex Systems and Dynamic Graphs. CoRR, abs/0803.2093, 2008.

[Th17] Then, M.; Kersten, T.; Günnemann, S.; Kemper, A.; Neumann, T.: Automatic Algorithm Transformation for Efficient Multi-Snapshot Analytics on Temporal Graphs. PVLDB, 10(8):877–888, 2017.

[vR04] van Raan, Anthony F. J.: Sleeping Beauties in science. Scientometrics, 59(3):467–472, Mar 2004.