B. König-Ries et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2023),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2023 1

Evolution of Degree Metrics in Large Temporal Graphs

Christopher Rost[1] Kevin Gomez[1] Peter Christen[2] Erhard Rahm[1]

Abstract: Graph metrics, such as the simple but popular vertex degree and others based on it, are well defined for static graphs. However, adapting static metrics for temporal graphs is still part of current research. In this paper, we propose a set of temporal extensions of four degree-dependent metrics, as well as aggregations like minimum, maximum, and average degree of (i) a vertex over a time interval and (ii) a graph at a specific point in time. We show why using the static degree can lead to wrong assumptions about the relevance of a vertex in a temporal graph and highlight the need to include *time* as a dimension in the metric. We propose a baseline algorithm to calculate the degree evolution of all vertices in a temporal graph and show its implementation in a distributed in-memory dataflow system. Using real-world and synthetic datasets containing up to 462 million vertices and 1.7 billion edges, we show the scalability of our algorithm on a distributed cluster achieving a speedup of around 12 on 16 machines.

Keywords: Temporal Property Graph; Temporal Degree; Degree Evolution; Temporal Graph Metric

1 Introduction

Temporal graphs are graphs that change in structure and content over time, where changes are captured and maintained as part of the graph data model. Many approaches exist to formally define a temporal graph [Iy21, Ko09, Ro22, HR21]. A graph's evolution is either represented as a series of snapshots, or by vertex and edge annotations for timestamps or time intervals describing their validity. These extended graph models allow analyzing the current or a past state of a graph as well as the evolution of the graph. Examples for temporal graph analysis are the exploration of human contact networks to detect the transmission of a disease [SK05, RKC01] or analyzing the change in the utilization of bike rental stations [Li15, Tl20]. In such graphs, the concepts of graph metrics also change because time is added as a new dimension. Metrics used for the characterization of static graphs need to be redefined or extended to take temporal evolution into account [Ni13].

One simple yet important metric of a vertex is the *vertex degree* [GY03]. It is determined by the number of incoming and outgoing edges (which is, except for multigraphs, equal to the number of neighbors) and thus a simple indicator for the relevance or importance of a vertex in a static graph. A vertex with a high degree can be seen as a strongly connected vertex, whereas a vertex with a degree of zero is an isolated vertex or singleton. The vertex degree is also known as the centrality measure *degree centrality* [Fr78], that can be used to find,

[1] University of Leipzig & ScaDS.AI Dresden/Leipzig, Germany. {rost,gomez,rahm}@informatik.uni-leipzig.de
[2] Australian National University, Canberra, Australia. peter.christen@anu.edu.au

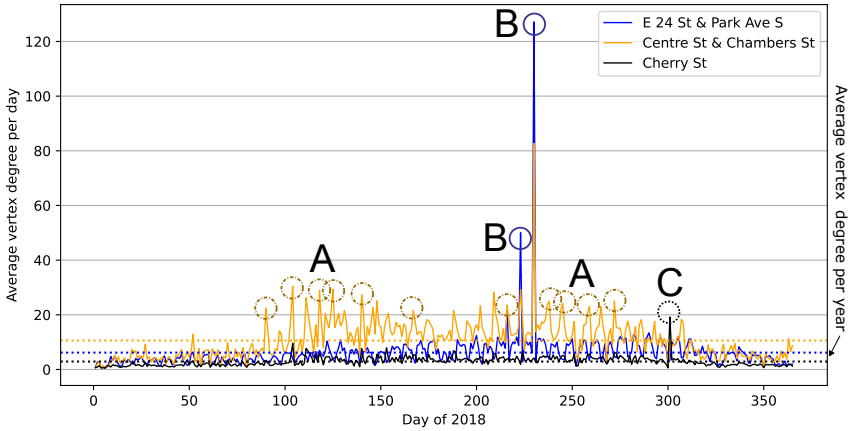


Fig. 1: Degree evolution of selected rental stations in NYC for 2018. For each day, the average degree is plotted. **A** indicates peaks on weekends, **B** a construction embargo event and **C** a Halloween parade.

for example, popular people according to their number of friendships in a social network, or the stations with the highest throughput of bike rentals in a bike-sharing network.

The minimum and maximum degrees are metrics that describe the vertices with the smallest and largest numbers of connections, respectively. The *degree range* [LJ21], *degree variance* [LJ21, Sn81, SE20] and the *average nearest neighbor degree* (ANND) [LJ21, YvdHL17], are aggregate metrics that can reveal important graph and vertex characteristics. The *degree range* of a graph (the difference between the maximum and minimum degree) describes the connectivity gap between the best and least connected vertices. For a bike-sharing network, a small degree range indicates a good distribution of rental stations without any hardly visited stations, whereas a high degree range indicates irregular usage. Another extended measure of a graph's heterogeneity is the *degree variance*, where a high variance shows a high inequality in the connectivity of the vertices. The *ANND*, on the other hand, reveals if a vertex is connected to others with a high connectivity, e.g., a social network user who is mainly friend with other users who are strongly connected.

Using only the static vertex degree is of limited value in an evolving graph as it cannot reflect the impact of topology changes. The same restriction applies for static aggregated metrics such as the average degree value [KA12] or the sum of all degrees [TBF17]. There is no information about *when* a vertex has what degree, *how long* this degree is valid, and *when* it increases or decreases. This is important, for example, in a bike-sharing network where vertices represent stations and directed edges connect the start and return stations of bike rentals.

Fig. 1 shows the time series representing the evolution of the vertex degree of three selected bike rental stations in NYC for 2018, calculated from the publicly available dataset also used

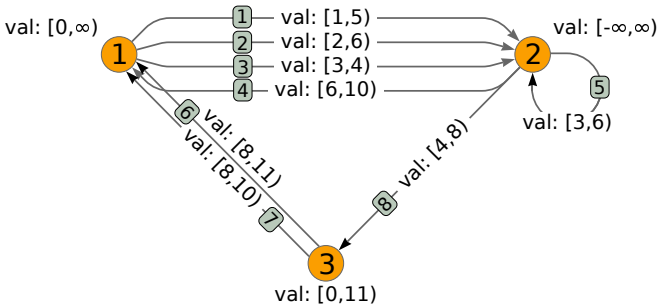


Fig. 2: An example temporal graph.

in our evaluations (see Sect. 6). For example, one can see the popularity of the station at Centre St & Chambers St on weekends by periodic peaks (marked with **A**) or the significantly higher rental rate of two stations during the Summer Streets Construction Embargo[3] in August (marked with **B**). Further, the impact of a Halloween parade[4] on Cherry St. (marked by **C**) is visible in these time-series. This shows that there are stations that are generally popular, such as in a city center or near train stations, as well as stations that are only popular at certain times, e.g., on weekends or during events. Further, comparing stations using the static or aggregated metrics, which are shown in Fig. 1 as dotted lines, may lead to the assumption that they seem equal by sharing a similar degree value, which in fact is not true over time which can be revealed by temporal metrics.

Fig. 2 shows a toy example of a temporal graph, which we use to illustrate the problem further. Each vertex and directed edge has a unique numeric identifier and a left-close right-open time interval $[\omega_a, \omega_b)$[5] assigned. For example, the edge with identifier 5 (hereinafter referred to as e_5) is valid from time point 3 (in the following denoted as ω_3) to ω_6, whereas the vertex v_1 is valid from ω_0 to the maximum upper bound, denoted by the infinity symbol ∞ (ω_{max}).

From a static perspective, if we disregard the graph's evolution, we can see that the vertex degrees are $deg(v_1) = 6$, $deg(v_2) = 7$, and $deg(v_3) = 3$. However, if time is considered, then the degree values change continuously so that the evolution of the degree value forms a time series. For example, at time ω_1, the degree of v_1 is 1, and the same at time ω_5. Further, since v_1 is valid until forever and the last validity of its edges end at time ω_{11} (exclusive), the degree from ω_{11} to forever (ω_{max}) is 0. Fig. 3 exemplifies the evolution of the degrees of v_1, v_2 and v_3, inclusive in- and outdegree of v_1 ($deg^-(v_1)$ and $deg^+(v_1)$).

It can be seen that the maximum degree of vertex v_1 is only 3 over its entire period of validity. From the vertex lower bound ω_0 to time ω_1, the degree is 0 – the same from ω_{11}

[3] https://www.milrose.com/insights/2018-summer-streets-construction-embargo (visited 2022-11-01)

[4] https://patch.com/new-york/east-village/halloween-dog-parade-2018-what-you-need-know (visited 2022-11-01)

[5] For simplicity we use integer interval bounds. It holds $[\omega_a, \omega_b) := \{\omega \in \mathbb{N} : \omega_a \leq \omega < \omega_b\}$.

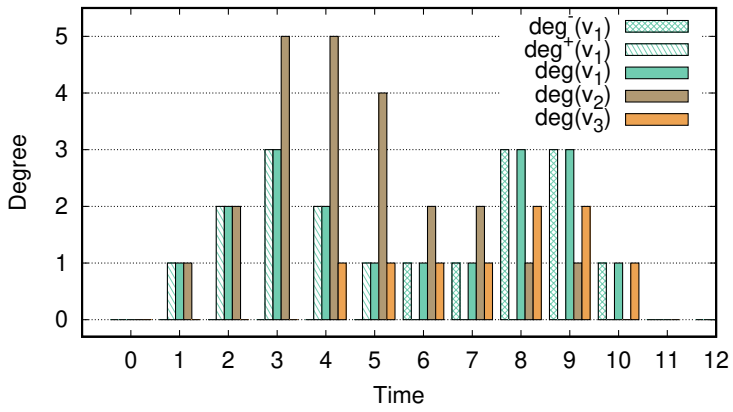


Fig. 3: Degree evolution of vertex v_1, v_2 and v_3 from ω_0 to ω_{12}. In addition, the indegree $deg^-(v_1)$ and outdegree $deg^+(v_1)$ are given for v_1.

onwards. Compared to the static point of view, where the degree is 6 for v_1, we can see that during the evolution of the graph the vertex never reaches this value. The same holds for the bike rental example of Fig. 1. For example, the static degree of the station "Cherry St." is 50, whereas the maximum value over the year is just 20 for a single day. This shows the importance of considering the changes of the degree metric over time. The use of the static degree metric for assessing the importance of a vertex can lead to misinterpretations, whereas using the degree evolution provides the exact degree for any time in the lifetime of the graph.

Contributions: In this work we focus on four time-sensitive degree-dependent graph measures: the vertex degree itself and its aggregations, the degree range, the degree variance, and the average nearest neighbor degree. We extend these well known static metrics with a time dimension and establish two new formal definitions per metric: (i) a temporal version which defines the metric at a specific point in time, and (ii) an evolutionary version which defines the change of the metric within a time interval as a time series. We then present a baseline algorithm that can calculate the degree evolution for all vertices of a given temporal graph. Using a binary search tree called *degree tree*, the algorithm efficiently maintains the degree changes of each vertex. We show how our algorithm can be adapted to a distributed processing model, which is further illustrated by the implementation as a graph analysis operator using a distributed in-memory dataflow system. In our experiments, we evaluate the scalability of our implementation which shows a sublinear growth of runtime by increasing dataset size as well as a speedup of up to 12 on a cluster with 16 physical machines.

2 Related work

Some works have defined a degree metric for vertices in a temporal graph, mainly by expanding the static version for temporal graphs. Thompson et al. [TBF17] introduce a *temporal degree centrality* metric for the domain of network neuroscience. They show that a node's influence in a temporal network can be represented by the centrality metric, which is the sum of the number of edges across a series of time points. If an edge is valid for multiple time points, it will be counted multiple times. However, this approach does not quantify the temporal order of edges so that different vertices with identical metrics cannot be distinguished.

A similar definition of temporal degree centrality is given by Long et al. [Lo20] and Wu et al. [Wu14]. Both calculate the sum of degrees over a time interval, which provides an estimate of a node's centrality in a temporal network. Wang et al. [Wa17] propose the *temporal degree deviation centrality* metric that can be calculated from a temporal network using graph snapshots. A similar approach defines the temporal degree as the number of nodes to which a vertex is linked in all timestamps of an interval without interruption [Ci20].

The *time-ordered graph* model by Kim et al. [KA12] can represent a dynamic network with a fixed vertex set and interval edges. For graphs of this model, several centrality metrics were introduced (including degree) to include the graph's temporal characteristic. Temporal degree is defined as the degree $D_{i,j}(v)$ for a vertex $v \in V$ in a time interval $[i, j]$. Tlebaldinova et al. [Tl20] use the degree as a temporal measure of centrality for bike-sharing stations. They show that the changing degree determines the time-distributed intensity of incoming and outgoing bike flows at a station.

In all these related works, the temporal degree is mostly seen as a scalar, aggregated (summed) value over a certain time interval, that is used as a centrality measure. In our approach, described next, we define both a *temporal degree* at a specific point in time as well as *degree evolution* for a time interval as a time series. This allows exact statements when a metric has what value for how long. In addition, our data model allows both changes in vertices and edges, as we describe in Sect. 3.1.

3 Degree-dependent metric evolution

We first define the temporal graph data model we use as a basis for our work, and then introduce new temporal notations of degree-dependent metrics for vertices in Sect. 3.2, and metrics for a whole temporal graph in Sect. 3.3.

3.1 Temporal graph model

We use a simplified version of the *Temporal Property Graph Model (TPGM)* data model [Ro22][6]. Although the model supports bitemporal versioning, for simplicity we limit ourselves to one time dimension. Thus, vertices and edges are assigned with a left-closed right-open time interval to represent the element's validity according to application-specific valid-time. Unlike most temporal graph models [Ca21], not only the edge set is dynamic, but the vertex set can also change over time. Contact sequence graphs [Ho18] can also be modeled by representing the time ω_i of the contact as time interval $[\omega_i, \infty)$, $[\omega_i, \omega_{i+1})$ or $[\omega_i, \omega_j)$ (depending on the use-case), where ω_j is the time of a subsequent contact. A TPGM graph is formally defined as follows:

Definition 1 (TPGM graph [Ro22, GS20]) *A TPGM graph is a directed multigraph* $\mathcal{G} = (V, E, \Omega)$ *with the following specifications:*

V is a finite set of vertices. Each vertex $v \in V$ is a tuple $\langle v_{id}, \tau \rangle$, where v_{id} is a unique vertex identifier, τ is a time-interval of the form $[\omega_{start}, \omega_{end})$ for which the vertex is valid with respect to Ω (defined below). We constrain that each $v \in V$ has at least one edge throughout the graph history, i.e., vertices that were isolated over the entire graph lifetime are not part of V.

E is a finite set of edges. Each edge $e \in E$ is a tuple $\langle e_{id}, s_{id}, t_{id}, \tau \rangle$, where e_{id} is a unique edge identifier that allows multiple edges between the same nodes, s_{id} and t_{id} are the source and target vertex identifier, τ is the time-interval of the form $[\omega_{start}, \omega_{end})$ for which the edge is valid with respect to Ω.

Ω represents the valid-time domain where an instant in time is a time point ω_i with limited precision, e.g., milliseconds. A time interval $\tau = [\omega_{start}, \omega_{end})$ with $\omega_{start}, \omega_{end} \in \Omega$ starts at ω_{start} and ends at ω_{end}. Since it is a left-close right-open interval, it includes ω_{start} but excludes ω_{end}.

We refer to our previous work [Ro22], in which several constraints are defined to ensure a consistent TPGM graph. Since the set of nodes V and edges E changes over time, we introduce two time-dependent sets of nodes and edges that we use later in the formal definitions in Sect. 3.2 and Sect. 3.3:

- $V(\omega_i) \subseteq V$ is a finite subset of vertices, where each vertex is valid at the given time point ω_i, i. e., for all $v = \langle v_{id}, \tau \rangle \in V(\omega_i)$ with $\tau = [\omega_{start}, \omega_{end})$ it holds: $\omega_{start} \leq \omega_i < \omega_{end}$.

[6] A TPGM graph, in addition, formally defines the concept of so-called logical graphs and assigns type labels and properties (key-value pairs) to nodes, edges, and logical graphs. Since neither is relevant for this work, we excluded it for the sake of simplicity.

- $E(\omega_i) \subseteq E$ is a finite subset of edges, where each edge is valid at the given time point ω_i, i. e., for all $e = \langle e_{id}, s_{id}, t_{id}, \tau \rangle \in E(\omega_i)$ with $\tau = [\omega_{start}, \omega_{end})$ it holds: $\omega_{start} \leq \omega_i < \omega_{end}$.
- $G(\omega) = (V(\omega), E(\omega))$ is a graph snapshot (or state) of a temporal graph G at a specific point in time ω.

3.2 Vertex-centric temporal degree metrics

For each of the following degree-based metrics, we first refer to the static version and then introduce our temporal and evolutionary version of the respective metric.

Vertex degree and aggregations. According to graph theory [GY03, Di10], the static (non-temporal) **vertex degree** $deg(v)$ is formally defined as follows:

Definition 2 (Vertex degree [GY03, Di10]) *The **degree** (or valence) of a vertex v in a static graph $G = (V, E)$, denoted $deg(v)$, is the number of proper edges incident to v plus twice the number of self-loops. Simplified, the degree of a vertex is the number of its edges. The **indegree** of a vertex v, denoted as $deg^-(v)$, is the number of edges directed to v whereas the **outdegree** of vertex v, denoted as $deg^+(v)$, is the number of edges directed from v. Each self-loop at v counts one toward the indegree of v and one toward the outdegree.*

Having a static view on the graph of Fig. 2, example vertex degrees are $deg(v_1) = 6$, $deg(v_2) = 7$, $deg^+(v_2) = 3$, and $deg^-(v_3) = 1$.

For temporal graphs, we now define the *temporal degree* as the degree of a vertex *at a specific point in time*.

Definition 3 (Temporal degree) *The **temporal degree** of a vertex v in a temporal graph $G = (V, E)$, denoted as $degt(v, \omega)$, is the degree of that vertex at time ω in the graph snapshot $G(\omega)$. It is defined as:*

$$degt(v, \omega) \begin{cases} deg(v), & \text{if } v \in V(\omega), \\ not\ defined, & otherwise. \end{cases} \tag{1}$$

*If $v \notin V(\omega)$, the degree is not defined. Analogous to the static degree, the **temporal indegree** $degt^-(v, \omega)$ is the number of edges directed to v, and **temporal outdegree** $degt^+(v, \omega)$ is the number of edges directed from v, at time ω.*

For example, in the graph of Fig. 2, the temporal degree of vertex v_1 at time ω_4 is $degt(v_1, \omega_4) = 2$, whereas the temporal indegree of vertex v_1 at time ω_8 is $degt^-(v_1, \omega_8) = 3$. There are clear differences between the static compared to the temporal metrics.

From the perspective of a vertex v, the degree of that vertex changes according to the existence of neighbours of v. For a given time interval τ, we thus define the *degree evolution* as a time series of temporal degrees, which contains all degree values with their corresponding time in the given interval.

Definition 4 (Degree evolution) *The **degree evolution** $degev(v, \tau) := \{x_1, x_2, ..., x_m\}$ of a vertex v is a time series of elements $x_i := degt(v, \omega)$, with $1 \leq i \leq m$ and $m = \omega_{end} - \omega_{start}$. Each x_i represents a temporal degree at time ω_j, i.e., x_1 at time point ω_{start} and x_m at $\omega_{end} - 1$, for the interval $\tau = [\omega_{start}, \omega_{end})$. Further, the temporal degree is a special case of the degree evolution: $degev(v, \tau) = \{degt(v, \omega_i)\}$ with $\tau = [\omega_i, \omega_{i+1})$ as an interval with a single time point. Furthermore, $degev^+(v, \tau)$ denotes the **outdegree evolution** whereas $degev^-(v, \tau)$ denotes the **indegree evolution**.*

For our example graph of Fig. 2, the degree evolution of vertex v_1 in the interval $\tau = [\omega_0, \omega_{11})$ is $degev(v_1, \tau) = \{0, 1, 2, 3, 2, 1, 1, 1, 3, 3, 1\}$.

The degree evolution defines the development of a vertex degree over a given time interval. This can now be used to determine the minimum, maximum and average degree of *a vertex* over a time interval, i.e., a vertex-centric aggregation.

Definition 5 (Vertex-centric min/max/avg degree) *The **vertex-centric minimum degree** of a vertex v within a time interval τ is the smallest value of all temporal degrees of v in this interval. Similarly, the **vertex-centric maximum degree** is the largest value and the **vertex-centric average degree** is the average value over all time points $\omega \in \tau$. With $|\tau|$ as the number of all time points in the interval τ holds:*

$$deg_{min}(v, \tau) := min\{degt(v, \omega) | \forall \omega \in \tau\}, \tag{2}$$

$$deg_{max}(v, \tau) := max\{degt(v, \omega) | \forall \omega \in \tau\}, \tag{3}$$

$$deg_{avg}(v, \tau) := \frac{1}{|\tau|} \sum_{\omega \in \tau} degt(v, \omega). \tag{4}$$

Average Nearest Neighbor Degree. An analyst may be interested in whether entities in a graph tend to connect to others with a high connectivity, or, the opposite case, connections occur randomly and irrespective of the degree [LJ21]. The former situation is referred to as *preferential attachment* in network science [JNB03] and applies to many real-world networks [Ne01, Ca06], including evolving networks [JNB03]. A metric to measure this tendency is the *average nearest neighbor degree* (ANND) $deg_{nn}(v)$. For a vertex v, the ANND is the sum of the direct neighbor degrees divided by the degree of v.

Definition 6 (Average nearest neighbor degree [LJ21]) *The **average nearest neighbor degree** $deg_{nn}(v_i)$ of a vertex v_i of a static graph G is defined as the sum of the degrees of each of the vertex' neighbor v_j divided by the degree of v_i:*

$$deg_{nn}(v_i) := \frac{1}{deg(v_i)} \sum_{v_j \in N(v_i)} deg(v_j). \tag{5}$$

The set $N(v_i) \subset V$ is defined as the set of vertices incident to a vertex v_i (its neighbors).

From a static perspective of the example graph of Fig. 2, the ANNDs are $deg_{nn}(v_1) = \frac{deg(v_2)+deg(v_3)}{deg(v_1)} = 1.67$, $deg_{nn}(v_2) = 1.43$ and $deg_{nn}(v_3) = 4.34$. These results suggest that vertex v_3 seems to have the strongest tendencies to connect to others who are also popular, while v_1 and v_2 display weaker tendencies. The average degree of the graph (here $deg_{avg} = 5.34$) can be used to interpret an ANND value. The larger the value compared to the average degree of the graph, the more likely we can assume that its neighbors are more popular than average. As the other degree-dependent metrics, the ANND will change over time if a graph evolves. To calculate the ANND of a vertex at a specific point in time, we now define the *temporal average nearest neighbour degree*:

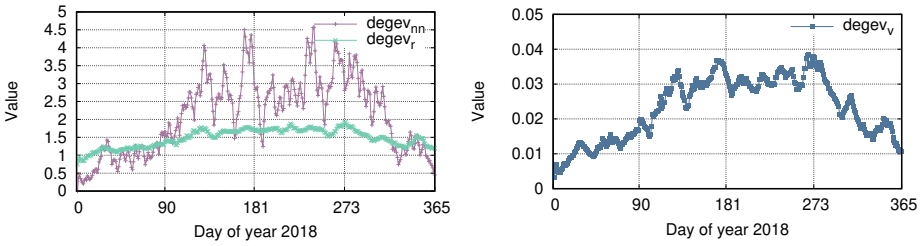
Definition 7 (Temporal ANND) *The **temporal average nearest neighbor degree** (TANND) $degt_{nn}(v_i, \omega)$ of a vertex v_i is defined as the sum of the temporal degrees of each of the vertex' neighbor (at time ω) divided by the temporal degree of v_i. Furthermore, the set $N(v_i, \omega) \subset V(\omega)$ is defined as the set of neighbors of vertex v_i at time ω. It then holds:*

$$degt_{nn}(v_i, \omega) := \frac{1}{degt(v_i, \omega)} \sum_{v_j \in N(v_i, \omega)} degt(v_j, \omega). \tag{6}$$

For the example in Fig. 2, the TANND for v_1 at time ω_4 is $degt_{nn}(v_1, \omega_4) = 2.5$, while at time ω_9 it is $degt_{nn}(v_1, \omega_9) = 1$.

An analyst may be also interested in the evolution of the ANND over a time interval, like how people's propensity to rent a bike from one popular location and ride to another popular location is changing within a month. We introduce the *average nearest neighbor degree evolution* to define a series of TANND values within a time interval.

Definition 8 (ANND evolution) *The **average nearest neighbor degree evolution** (ANNDE) $degev_{nn}(v, \tau) := \{x_1, x_2, ..., x_m\}$ of a vertex v is a time series of elements $x_i := degt_{nn}(v, \omega)$, with $1 \leq i \leq m$ and $m = \omega_{end} - \omega_{start}$. Each x_i represents the TANND at time ω_j, i.e., x_1 at time point ω_{start} and x_m at $\omega_{end} - 1$, for the interval $\tau = [\omega_{start}, \omega_{end})$. The TANND is a special case of the ANNDE: $degev_{nn}(v, \tau) = \{degt_{nn}(v, \omega)\}$, with $\tau = [\omega_i, \omega_{i+1})$ as an interval with a single time point.*



(a) ANNDE and degree range evolution.

(b) Degree variance evolution.

Fig. 4: Resulting time-series of selected degree evolution metrics of dataset citibike for year 2018.

For example, the ANNDE of v_1 in the interval $\tau = [\omega_1, \omega_5)$ is $degev_{nn}(v_1, \tau) = \{1, 1, 2.67, 2\}$. For our small example graph, the ANND remains quite small in this interval, which means that the popularity of the neighbours of v_1 does not increase much. Fig. 4a shows the resulting ANNDE time series of a selected rental station for the real world bike-sharing graph we are using in our evaluation in Sect. 6. One can see that the tendency that rentals happen between popular stations are high during the summer months.

3.3 Graph-centric temporal degree metrics

After looking at metrics for individual vertices of a graph, we now develop metrics that concern an entire graph. Several metrics have already been defined for aggregating all vertices of a static graph, such as the *minimum*, *maximum*, and *average degree*.

Definition 9 (Min/max/avg degree of a graph [LJ21]) *The **minimum**, **maximum**, and **average degree** of a static graph G are defined as the minimum, maximum, and average value of all vertex degrees $deg(v)$ for all $v \in V$. It holds:*

$$deg_{min}(G) := min\{deg(v) | v \in V\}, \tag{7}$$

$$deg_{max}(G) := max\{deg(v) | v \in V\}, \tag{8}$$

$$deg_{avg}(G) := \frac{1}{|V|} \sum_{v \in V} deg(v), \tag{9}$$

with $deg_{min}(G) \leq deg_{avg}(G) \leq deg_{max}(G)$.

For the example graph in Fig. 2, the minimum, maximum, and average degrees are $deg_{min}(G) = 3$, $deg_{max}(G) = 7$ and $deg_{avg}(G) = 5.34$.

With the evolution of a graph, any aggregated graph metric can change over time. We therefore define the *minimum*, *maximum*, and *average temporal degree* as an aggregated value of all vertices $V(\omega)$ in a temporal graph at time ω.

Definition 10 (Min/max/avg temporal degree) *The **minimum**, **maximum** and **average temporal degree** of a temporal graph G are the minimum, maximum and average values of all temporal vertex degrees at time ω. With $V(\omega)$ as the set of vertices at time ω it holds:*

$$degt_{min}(G,\omega) := min\{degt(v,\omega)|v \in V(\omega)\}, \tag{10}$$

$$degt_{max}(G,\omega) := max\{degt(v,\omega)|v \in V(\omega)\}, \tag{11}$$

$$degt_{avg}(G,\omega) := \frac{1}{|V(\omega)|} \sum_{v \in V(\omega)} degt(v,\omega), \tag{12}$$

with $degt_{min}(G,\omega) \leq degt_{avg}(G,\omega) \leq degt_{max}(G,\omega)$.

For the example graph in Fig. 2, at time ω_4, the aggregated degrees are $degt_{min}(G,\omega_4) = 1$, $degt_{max}(G,\omega_4) = 5$ and $degt_{avg}(G,\omega_4) = 2.67$.

Degree range. The minimum degree reveals the smallest set of connections of a graph's vertices, whereas the maximum degree gives a measure of the most connections an vertex has in the graph. The difference between the minimum and maximum degree of any vertex in a graph is called the *degree range* [LJ21]. It provides a measure of the heterogeneity (or gap) between the connectivity of the most and the least connected vertices in a graph [LJ21].

Definition 11 (Degree range [LJ21]) *The **degree range** of a static graph $G = (V,E)$, denoted as $deg_r(G)$, is the difference between the maximum and minimum degree:*

$$deg_r(G) = deg_{max}(G) - deg_{min}(G). \tag{13}$$

From a static view on the example graph of Fig. 2, the degree range is $deg_r(G) = 7 - 3 = 4$, which suggests that it has a high inequality related to connectivity. Now considering a temporal graph, the *temporal degree range* provides information about the degree range of a graph at a specific point in time.

Definition 12 (Temporal degree range) *The **temporal degree range** $degt_r(G,\omega)$ of a temporal graph G at time ω is defined as the difference between the maximum and minimum temporal degree:*

$$degt_r(G,\omega) = degt_{max}(G,\omega) - degt_{min}(G,\omega). \tag{14}$$

With respect to the example graph from Fig. 2, the temporal degree range at time ω_4 is $degt_r(G,\omega_4) = 5 - 1 = 4$, which is equal to the static metric, while at times ω_1, ω_6 and ω_{10}, the temporal degree range is $degt_r(G,\omega_1) = degt_r(G,\omega_6) = degt_r(G,\omega_{10}) = 1$. Thus, as the graph evolves, the degree range changes as well.

To obtain any changes of the degree range over a defined time interval, we introduce the *degree range evolution* that defines a series of temporal degree range values for all time points in a given interval.

Definition 13 (Degree range evolution) *The **degree range evolution** $degev_r(G, \tau) := \{x_1, x_2, ..., x_m\}$ of a temporal graph G is a time series of elements $x_i := degt_r(G, \omega)$, with $1 \leq i \leq m$ and $m = \omega_{end} - \omega_{start}$. Each x_i represents the temporal degree range at time ω_j, i.e., x_1 at time point ω_{start} and x_m at $\omega_{end} - 1$, for the interval $\tau = [\omega_{start}, \omega_{end})$. The temporal degree range is a special case of the degree range evolution: $degev_r(G, \tau) := \{degt_r(G, \omega)\}$, with $\tau = [\omega_i, \omega_{i+1})$ as an interval with a single time point.*

For the example graph of Fig. 2, the degree range evolution for $\tau = [\omega_0, \omega_7)$ is $degev_r(G, \tau) = \{0, 1, 2, 5, 4, 3, 1\}$, which shows a changing gap of connectivity in this interval. Fig. 4a shows the time series of the degree range evolution for the real world bike sharing graph we are using in our evaluations. One can see that the value is below 2 over the whole year which indicates a low inequality of rentals between all rental stations.

Degree variance. Besides the simple metric of range, Snijders introduced the more complex metric called *degree variance* of a graph [Sn81], which involves its average degree to characterize the *heterogeneity* in connectivity across nodes. This metric reveals information about the spread of both well-connected and not so well-connected vertices in a graph. It is formally defined as follows:

Definition 14 (Degree variance [LJ21]) *The **degree variance** $deg_v(G)$ of a graph G is defined as the sum of the square of the difference between each vertex degree $deg(v)$ and the average degree of the graph $deg_{avg}(G)$, divided by the total number of vertices $|V|$:*

$$deg_v(G) := \frac{\sum_i (deg(v) - deg_{avg}(G))^2}{|V|}. \tag{15}$$

This metric quantifies the extent to which there are differences in the connectivity of the vertices in a graph. High differences in connectivity mean high variance; if all node degrees are the same then the degree variance is zero. If the example graph in Fig. 2 is considered static it has a degree variance of $deg_v(G) = 2.89$.

For temporal graphs, the degree of vertices can change over time, and so can the average degree as well as the number of vertices. Therefore, we formally define the *temporal degree variance* as follows:

Definition 15 (Temporal degree variance) *The **temporal degree variance**, $degt_v(G, \omega)$, of a temporal graph G is defined as the sum of the square of the difference between each temporal vertex degree $degt(v, \omega)$ and the temporal average degree of the graph $degt_{avg}(G, \omega)$ at time ω, divided by the total number of vertices $|V(\omega)|$ at that time:*

$$degt_v(G, \omega) := \frac{\sum_i (degt(v, \omega) - degt_{avg}(G, \omega))^2}{|V(\omega)|}. \tag{16}$$

Considering the example graph in Fig. 2 at ω_4, the temporal degree variance is $degt_v(G, \omega_4) = 2.89$, which is equal to the static value since the inequality of connectivity is the same for this small example. In contrast, at time ω_1, the temporal degree variance is $degt_v(G, \omega_1) = 0.22$ since there is a quite high equality of connectivity at this time. To evaluate whether and how the degree variance changes in a given time interval, i.e., if the inequality of degrees in a graph decreases or increases over time, or if it retains a similar value, we define the *degree variance evolution*.

Definition 16 (Degree variance evolution) *The **degree variance evolution** $degev_v(G, \tau) := \{x_1, x_2, ..., x_m\}$ of a temporal graph G is a time series of elements $x_i := degt_v(G, \omega)$, with $1 \leq i \leq m$ and $m = \omega_{end} - \omega_{start}$. Each x_i represents the temporal degree variance at time ω_j, i.e., x_1 at time point ω_{start} and x_m at $\omega_{end} - 1$, for the interval $\tau = [\omega_{start}, \omega_{end})$. Further, the temporal degree variance is a special case of the degree variance evolution: $degev_v(G, \tau) = \{degt_v(G, \omega_i)\}$ with $\tau = [\omega_i, \omega_{i+1})$ as an interval with a single time point.*

The degree variance evolution of vertex v_1 in the example graph of Fig. 2, for time interval $\tau = [\omega_0, \omega_5)$, is the series: $degev_v(G, \tau) = \{0, 0.22, 0.89, 2.89, 2.89\}$. The degree variance increases over time in this example, which indicates a growth of the inequality of the vertex' connectivity. With regard to the real world bike sharing graph, Fig. 4b shows the degree variance evolution of the temporal graph. The inequality of the rental stations' utilization is low over the whole year but reaches its lowest values in the winter months.

4 Degree evolution algorithm

We now describe a baseline algorithm that calculates the degree evolution (see Definition 4) for all vertices in a temporal graph.

We assume that the input is a temporal graph $G = (V, E)$ including a set of temporal vertices V and temporal edges E according to the TPGM model described in Sect. 3.1, where the degree type $\Psi = \{in, out, both\}$ is given as configuration parameter. The output of the algorithm is a time series representing the degree evolution for each vertex, where we reduce the size of the result by merging succeeding time points without a degree change into intervals. These intervals are tuples $\langle v_{id}, \tau_i, degt(v_{id}, \omega_j)\rangle$, where v_{id} is a vertex identifier, τ_i is the interval in which the degree is valid without interruption, and $degt(v_{id}, \omega_i)$ the constant temporal degree of v_{id} for any time point ω_j of the interval τ_i. We split the algorithm into five steps which we described next.

(1) Vertex mapping. For each vertex $v \in V$ we extract the vertex identifier and its time interval into a tuple $\langle v_{id}, \omega_{start}, \omega_{end}\rangle$. This tuple is later used as input of step (5). This step can be skipped if the vertex times are not of relevance. Considering our example graph

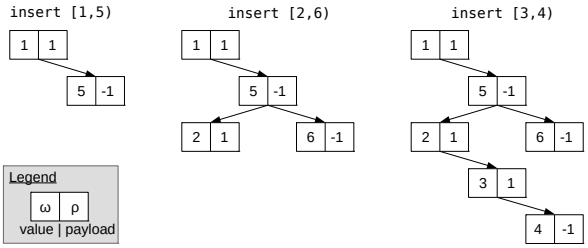


Fig. 5: Degree tree building for vertex v_1 and $\Psi = out$.

in Fig. 2, each of the graph's vertices $V = \{v_1, v_2, v_3\}$ is mapped to a tuple, resulting in a set of three tuples $\langle v_1, 0, \infty\rangle$, $\langle v_2, -\infty, \infty\rangle$ and $\langle v_3, 0, 11\rangle$[7].

(2) Edge mapping. For each edge $e \in E$ we extract the required vertex identifiers and the edge's time interval into one or two tuples $\langle v_{id}, \omega_{start}, \omega_{end}\rangle$ depending on the degree type Ψ. For $\Psi = in$, one tuple is created with $v_{id} \leftarrow t_{id}$ (the target vertex identifier), for $\Psi = out$ one tuple is created with $v_{id} \leftarrow s_{id}$ (the source vertex identifier), and for $\Psi = both$ both of these tuples are created. Considering the example graph in Fig. 2 and $\Psi = out$, each of the graphs edges $E = \{e_1, e_2, ..., e_8\}$ is mapped to one tuple as described above. For example, edge e_4 is mapped to $\langle v_2, 6, 10\rangle$, whereas e_5 is mapped to $\langle v_2, 3, 6\rangle$.

(3) Interval collection. We group the set of tuples $\langle v_{id}, \omega_{start}, \omega_{end}\rangle$ from step (2) by vertex identifier and create a mapping $v_{id} \rightarrow I_{v_{id}} = \{\tau_0, \tau_1, ..., \tau_n\}$ which assigns a unsorted set of edge intervals $I_{v_{id}}$ to the corresponding vertex identifier. For vertex v_1 and $\Psi = out$ of our example, the mapping to the collection of all incident edge intervals is $v_1 \rightarrow I_{v_1} = \{[1, 5), [2, 6), [3, 4)\}$.

(4) Capture degree evolution. For each vertex v and its corresponding unsorted set of (incoming, outgoing, or both) edge intervals created in step (3), a data structure maintaining the rise or fall of the metric at all respective points in time, i.e., when the degree of the vertex changes, is needed. A baseline approach is the maintenance of a typed list holding two types of points in time: the lower interval bounds which indicate a degree rise of 1, and the upper interval bounds which indicate a fall of 1. The space complexity is always $O(n)$ with $n = 2 \cdot |I_{v_{id}}|$, i.e., the number of all time points including duplicates. All points in time can be inserted with a time complexity $O(n)$ ($O(1)$ each), and the list has to be sorted before the iteration which costs $O(n \cdot log(n))$. The degree evolution for this vertex can be created by iterating the list (with $O(n)$) and adding 1 to a aggregate value for all lower interval bounds and -1 for all upper bounds.

An alternative is a Binary Search Tree (BST) [Be75] T_v. Each node of the tree has a value $\omega \in \Omega$ and a payload $\rho \in \mathbb{Z}$. ω represents a point in time, whereas ρ (initialized with 0) stores an aggregated value indicating the quantity of change (positive or negative) of the

[7]Note that we use integers for time points to improve readability.

degree at this specific time ω compared to the aggregated value of the evolution until this point in time. For a left-close right-open interval $\tau = [\omega_{start}, \omega_{end})$, the payload ρ of node ω_{start} is increased by 1, whereas ρ of ω_{end} is decreased by 1. Further, the left child node ω_l of a parent node ω_p has a value $\omega_l < \omega_p$ and the right child node ω_r has a value $\omega_r > \omega_p$, respectively. The worst case space complexity is $O(n)$, too, but having n without duplicate time points. The time complexity of inserting a node in this tree is $O(log(n))$ on average ($O(n)$ if all time points are different). The random insertion of points in time, while keeping the tree sorted, and the lower memory requirements by avoiding duplicated points in time, is our reason for choosing the BST, which will be called *degree tree* in the following. Thus, the output of this step (4) is a mapping $v_{id} \rightarrow T_v$ that assigns a degree tree to its corresponding vertex identifier.

If we again consider v_1 in our example, the building of the degree tree T_{v_1} assuming $\Psi = out$ is shown in Fig. 5. Inserting the interval $[1, 5)$ first inserts a node with value $\omega = 1$ and payload $\rho = 1$, and then a node with $\omega = 5$ and payload $\rho = -1$. For the subsequent two intervals, four additional nodes are added. A degree tree with six nodes is the result, as shown on the right side.

(5) Tree traversal and result collection For each vertex, we now have a degree tree T_v that represents the degree evolution of this vertex for the degree type Ψ, and the lower and upper bounds of the vertex' validity interval, ω_{start} and ω_{end}. If the validity of the vertices can be neglected, a default minimum and maximum time point can be used as initial values. Each degree tree is now traversed using Depth First Search (DFS) [Ta72] and in-order traversal (LNR) starting at the root node to obtain an ascending order of points in time. Algorithm 1 outlines this step.

The algorithm starts by traversing the tree T_v in line 5 with the recursive function InOrderDFS (lines 8 to 11). Function ProcessNode describes the logic of a node visit, where we first handle the special case of an vertex lower interval bound that is equal to the value of first visited node of the tree (lines 13 to 15). For every following visited node, the resulting temporal degree tuple is collected in line 17 if payload $\rho \neq 0$.

Next, to get the degree for the subsequent interval, the payload ρ is first added to d (line 18), and second the time point ω is remembered as lower interval bound for the next interval (line 19). After all nodes of the tree are visited, we check for a remaining time interval from the last time point ω_{last} to the vertex upper interval bound ω_{max} and collect a last tuple with $d = 0$ accordingly (line 7). The final algorithm output is a series of tuples $\langle v_{id}, \tau, degt(v_{id}, \omega)\rangle$, with $degt(v_{id}, \omega)$ as constant temporal degree for all time points $\omega \in \tau$, that were collected by both *collect()* calls (lines 7 and 17).

For a better understanding, we exemplary go through Algorithm 1 by using the degree tree T_{v_1} of vertex v_1, shown on the right side in Figure 5, as input. Remember this is the representation of the outdegree of v_1. In addition, from step (1), the algorithm gets the lower bound $\omega_{start} = 0$ and upper bound $\omega_{end} = \infty$ of the vertex interval as input

Algorithm 1: Tree traversal and result collection

```
   Data: T_v, v_id, ω_start, ω_end;                                  /* Input data */
1  ω_last ← ω_start ;                                  /* ω_start = −∞ if not given */
2  ω_max ← ω_end ;                                       /* ω_end = ∞ if not given */
3  d ← 0 ;                                            /* Initialize degree with 0 */
4  Function Main():
5  |  InOrderDFS(T_v);                          /* Traverse the tree with in-order DFS */
6  |  if ω_last < ω_max then                      /* Check for last remaining interval */
7  |  |  collect(⟨v_id, [ω_last, ω_max), d⟩);     /* Collect tuple for last interval */
8  Function InOrderDFS(tree):
9  |  if tree.left ≠ null then InOrderDFS(tree.left);
10 |  ProcessNode(tree.value,tree.payload);
11 |  if tree.right ≠ null then InOrderDFS(tree.right);
12 Function ProcessNode(ω, ρ):
13 |  if ω_last == ω then                                 /* Check first node visit */
14 |  |  d ← d + ρ;                                      /* Add payload to degree */
15 |  |  return ;                                              /* Leave function */
16 |  if ρ ≠ 0 then                                  /* Check if the degree changes */
17 |  |  collect(⟨v_id, [ω_last, ω), d⟩) ;                        /* Collect tuple */
18 |  |  d ← d + ρ ;                                /* Add degree change to degree */
19 |  |  ω_last ← ω;                                  /* Remember ω for next call */
```

parameters to initialize ω_{last} and ω_{max}. During the in-order traversal of the DFS, function `ProcessNode`(ω, ρ) is called first with the arguments $(1, 1)$ (value,payload), followed by $(2, 1)$, $(3, 1)$, $(4, -1)$, $(5, -1)$ and $(6, -1)$.

According to the first tuple, the interval $[0, 1)$ is defined and collected as part of the first resulting temporal degree tuple $\langle v_1, [0, 1), 0\rangle$ afterwards (line 17). Then, the payload 1 is added to the degree value d (line 18) and the timestamp value 1 is remembered in variable ω_{last} (line 19). In the next function call with input tuple $(2, 1)$, an interval $\tau \leftarrow [1, 2)$ is defined and collected together with the current degree value of d which is 1. The collected result tuple is thus $\langle v_1, [1, 2), 1\rangle$. Again, the degree value is updated by the payload and the timestamp is remembered. For the remaining four input tuples $(3, 1)$, $(4, -1)$, $(5, -1)$ and $(6, -1)$ will be the following result tuples collected: $\langle v_1, [2, 3), 2\rangle$, $\langle v_1, [3, 4), 3\rangle$, $\langle v_1, [4, 5), 2\rangle$ and $\langle v_1, [5, 6), 1\rangle$.

To collect also the remaining interval from 6 to ∞, the condition (line 7) checks whether the largest timestamp in the tree (ω_{last}) is smaller than the maximum timestamp ($\omega_{max} = \infty$). Since this is true in our case, we define the remaining interval $\tau = [6, \infty)$ and collect the output tuple $\langle v_1, [6, \infty), 0\rangle$ which states that the degree of v_1 is 0 for the interval $[6, \infty)$. The result of this final step is a compact representation of the *degree evolution* of the outdegree of vertex v_1 as defined by Definition 4: $degev^+(v_1, [0, \infty)) = \{\langle 0, [0, 1)\rangle, \langle 1, [1, 2)\rangle, \langle 2, [2, 3)\rangle, \langle 3, [3, 4)\rangle, \langle 2, [4, 5)\rangle \langle 1, [5, 6)\rangle \langle 0, [6, \infty)\rangle\}$.

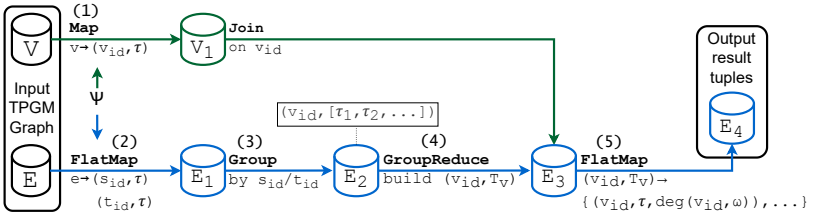


Fig. 6: Implementation details of the Degree Evolution-Operator.

5 Distributed implementation

The ability to process very large graphs efficiently is often a limitation of existing graph processing systems [Sa20], requiring partitioning of large graphs and distributed processing for example of analytical tasks. There are distributed graph processing systems, such as Tegra [Iy21] based on Apache Spark [Za16], or GRADOOP [Ro22, Ro21] which uses Apache Flink [Ca15]. An analytical operator in GRADOOP is a smart combination of Flink transformations. A Flink transformation, e.g., *map*, *flatMap* and *join*, is a processing unit that can be applied in parallel on a distributed Flink DataSet. A DataSet represents a distributed collection of elements of the same type in Apache Flink. Its tuples are distributed among all nodes of a cluster according to a partitioning strategy. We use this operator concept for our distributed implementation of the algorithm described in Sect. 4.

Fig. 6 shows an architectural sketch of a *Degree Evolution-Operator*[8] as a Directed Acyclic Graph (DAG) representing multiple Flink transformations that are applied on the input graph DataSets: V with $v_i = \langle v_{id}, \tau \rangle$ and E with $e_i = \langle e_{id}, s_{id}, t_{id}, \tau \rangle$. The enumeration of the data flow follows the algorithm steps given in Sect. 4.

First, in step (1), each vertex of the input vertex DataSet V, is mapped to a minimal representation holding the vertex identifier and the bounds of the vertex' time interval. The resulting DataSet is named V_1 in the figure. If the temporal information of the vertices can be neglected, this step can be skipped and default min/max timestamps can be used as input to step (5), which avoids the later described distributed join. Then, we apply a *FlatMap* transformation, step (2), to the edge DataSet E that is configured by the degree type ($\Psi \in \{in, out, both\}$) as selected by the user. According to the degree type, one or two tuples of the format $\langle v_{id}, \omega_{start}, \omega_{end} \rangle$ are extracted from an input edge tuple (step (2) in Sect. 4). The resulting DataSet is denoted as E_1.

On E_1, we apply a *Group* transformation which groups all entities by the vertex identifier, and creates a set of tuples $\langle \omega_{start}, \omega_{end} \rangle$ for each group. In the figure, this step is marked by (3), whereas the resulting grouped DataSet is denoted as E_2. Due to the grouping, E_2 is partitioned by the vertex identifier. For each group, we apply a *GroupReduce* transformation

[8]The operator code is open-source: `https://github.com/dbs-leipzig/gradoop/tree/develop/gradoop-temporal/src/main/java/org/gradoop/temporal/model/impl/operators/metric`.

	$\|V\|$	$\|E\|$	Size (GB)	$\sum \|degev()\|$
LDBC SF1	3.2 M	17.3 M	4.2	30.6 M
LDBC SF10	30.0 M	176.6 M	42.3	319.6 M
LDBC SF100	282.6 M	1.77 B	421.9	3.18 B
Citi Bike	1174	97.5 M	22.6	381.0 M
Stackoverflow	462.9 M	664.8 M	199.0	1.3 B

Tab. 1: Dataset statistics, including their sizes on HDFS and number of result set tuples for $\Psi = both$, i. e., $\sum_{i=1}^{|V|} |degev(v_i)|$. For example, 3.18B tuples result for the LDBC dataset with SF 100.

in step (4) which calls a user-defined function for each group. This function receives the whole group at once and produces a mapping $v_{id} \rightarrow T_v$ assigning a degree tree to its corresponding vertex identifier, represented as a tuple $\langle v_{id}, T_v \rangle$. The resulting tuples are part of DataSet E_3, which is partitioned by the vertex identifier.

Now, each tuple of V_1 needs to be joined by the vertex identifier to it's corresponding degree tree tuple of DataSet E_3 to extend it with the interval bounds of the vertex. As said before, this step can be optionally skipped. As a result of the join, the DataSet E_3 consists of tuples $\langle v_{id}, T_v, \omega_{start}, \omega_{end} \rangle$. As a last step, annotated with a (5), a *FlatMap* transformation is applied on DataSet E_3 where its internal logic implements the tree traversal and result collection process defined in Algorithm 1. For each input tuple, the transformation produces multiple (at least one) result tuples in the form $\langle v_{id}, \tau, degt(v_{id}, \omega) \rangle$, describing the constant temporal degree (see Definition 3) of vertex $v \in V$ (identified by v_{id}) for the whole interval τ. The resulting DataSet is named E_4.

6 Experimental Evaluation

We now evaluate the runtime and scalability of the temporal degree operator we discussed in Sect. 5 with respect to increasing data set and cluster sizes. We ran all experiments on a cluster with 16 worker nodes connected via 1 GBit Ethernet, where each worker consists of a E5-2430 6(12) 2.5 Ghz CPU, 48 GB RAM, two 4 TB SATA disks, and running openSuse 13.2, Hadoop 2.7.3 and Flink 1.9.0. On a worker node, a Flink Task Manager [Ca15] is configured with 6 task slots and 40GB memory.

We use three datasets for the evaluation, referred to as *LDBC* [Io16] (a synthetic social network in three scale factors), *citibike*[9] and *stackoverflow*[10] (both real-world data). In Fig. 4 we show example time series of four evolution metrics for the citibike dataset. Each graph is stored distributed using the Hadoop Distributed File System (HDFS) by hash partitioning as two datasets V and E. Table 1 shows statistics of the three datasets with the different scaling factors (SF) for LDBC. Each experiment includes reading the graph dataset from

[9] `https://www.citibikenyc.com/system-data/` (visited 2022-10-01).
[10] `https://archive.org/details/stackexchange` (visited 2022-10-01).

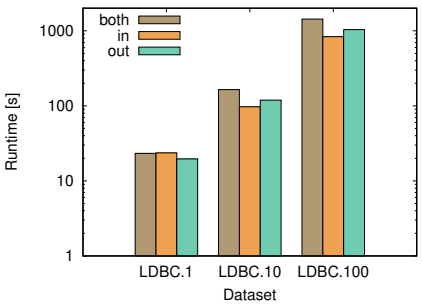


Fig. 7: Runtimes for linearly growing dataset sizes.

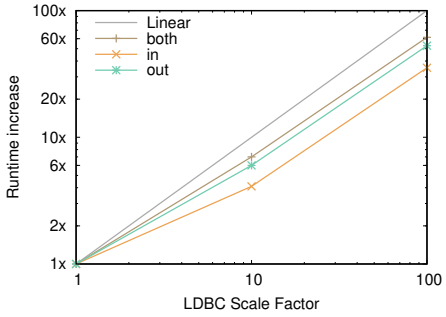


Fig. 8: Factor of runtime increase for linearly growing dataset sizes.

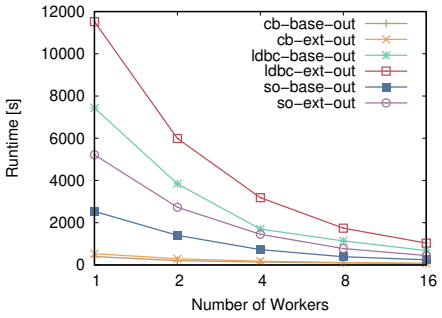


Fig. 9: Runtimes for #workers with $\Psi = out$.

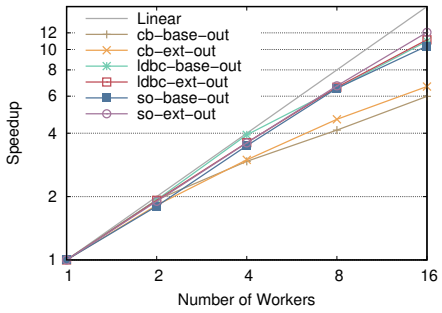


Fig. 10: Speedup of algorithm for $\Psi = out$.

the HDFS, executing the specific workflow, and finally writing all results back to the HDFS. We ran each experiment five times and report average runtimes.

Impact of dataset size. Fig. 7 and 8 show the impact of the dataset size to the operator runtime with full parallelism of 16 workers with respect to different degree types $\Psi \in \{in, out, both\}$. While Fig. 7 shows the actual runtime in seconds for all three dataset sizes, Fig. 8 visualizes the factor by which the runtime has increased compared to the runtime of the LDBC SF1 dataset. For example, the runtime for the LDBC SF1 dataset for $\Psi = both$ is only 23.3 seconds, for LDBC SF10 164.6 seconds (factor 7 higher compared to LDBC SF1) and for LDBC SF100 1433.6 seconds (factor 61). The best result is given by degree type $\Psi = in$, where the runtimes of LDBC SF100 are only 35.4 times larger compared to LDBC SF1, although the dataset is 100 times larger. From LDBC SF10 to LDBC SF100 the runtimes of all three degree types rise equally.

The results, specifically Fig. 8, show that a linear increase of the dataset size leads to only a sublinear increase in the running time for a constant graph structure. Further, the runtimes

of $\Psi = both$ are always higher compared to the others which is due to the double amount of collected tuples in step (2), as we discussed in Sect. 5.

Impact of worker count. We next examine the runtime and scalability of the algorithm for all datasets. In addition, the effect of excluding the vertex time information as described in Sect. 4 is evaluated. Without using the vertex time, the complete step (1) and the expensive join after step (4) can be avoided (see Sect. 5). In the following, we refer to an execution without vertex time as *base* and *extended* for the full algorithm. The results in Fig. 9 show that the mentioned higher complexity has a significant impact on the running time. For example, the runtime on a single machine for the citibike dataset is 397.6 seconds (base) and 533.6 seconds (extended), which means an increase of 34.2%. For the stackoverflow dataset, the execution takes 2,536 seconds (base) and 5,216 seconds (extended), which means almost doubling the runtime on a single machine.

The more workers are added, the smaller the runtime and the difference between the two algorithm variants, which can be seen in Fig. 10. With the citibike dataset, we can see that the runtimes on a single machine are already low and that only a moderate improvement can be achieved through horizontal scaling of resources. For this dataset, we reach a speedup of about 6.7 for 16 machines using the extended variant, while for the LDBC SF100 and stackoverflow datasets we achieve a speedup of up-to 11.1 and 12.07, respectively.

7 Conclusion

Most graphs that model real-world entities and their relationships are dynamic, where edges and vertices can be valid for only a certain period of time. One simple but often used centrality measure is the degree centrality using a vertex' degree to judge it's popularity in a network. We show in this work that it is necessary to determine a vertex degree over time, to know exactly *when* a node has which degree and *how long* this value is valid and in which quantity it does change over time. We therefore provide temporal extensions to the vertex degree metric itself, its aggregations and others based on it, namely the degree range, the degree variance and the ANND, and define them formally. We further describe an algorithm to calculate the newly introduced degree evolution for all vertices of a temporal graph. We implemented the algorithm as a graph analysis operator in GRADOOP [Ro22], an open-source distributed graph analysis system.

We evaluated runtimes and scalability of the operator on a cluster with 16 machines to determine the impact of different datasets and sizes. In summary, we have shown that a linear increase in the dataset size leads to only a sublinear increase in runtime of our algorithm. We also showed that the operator scales well by increasing the number of machines. Speedup values between 10 and 12 were achieved on 16 machines using the two largest datasets.

Acknowledgement. The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany and by the Sächsische Staatsministerium für Wissenschaft, Kultur und Tourismus for ScaDS.AI.

Bibliography

[Be75] Bentley, Jon Louis: Multidimensional Binary Search Trees Used for Associative Searching. Commun. ACM, 18(9):509–517, sep 1975.

[Ca06] Capocci, Andrea et al.: Preferential attachment in the growth of social networks: The internet encyclopedia Wikipedia. Physical review E, 74(3):036116, 2006.

[Ca15] Carbone, Paris et al.: Apache Flink: Stream and Batch Processing in a Single Engine. IEEE Data Eng. Bull., 38(4):28–38, 2015.

[Ca21] Casteigts, Arnaud; Meeks, Kitty; Mertzios, George B.; Niedermeier, Rolf: Temporal Graphs: Structure, Algorithms, Applications (Dagstuhl Seminar 21171). Dagstuhl Reports, 11(3):16–46, 2021.

[Ci20] Ciaperoni, Martino; Galimberti, Edoardo; Bonchi, Francesco; Cattuto, Ciro; Gullo, Francesco; Barrat, Alain: Relevance of temporal cores for epidemic spread in temporal networks. Scientific reports, 10(1):1–15, 2020.

[Di10] Diestel, Reinhard: Graph Theory, 4th Edition. Springer, 2010.

[Fr78] Freeman, Linton C: Centrality in social networks conceptual clarification. Social Networks, 1(3):215–239, 1978.

[GS20] Gandhi, Swapnil; Simmhan, Yogesh: An interval-centric model for distributed computing over temporal graphs. In: IEEE 36th International Conference on Data Engineering (ICDE). pp. 1129–1140, 2020.

[GY03] Gross, Jonathan L; Yellen, Jay: Handbook of graph theory. CRC press, 2003.

[Ho18] Holme, Petter: Temporal Networks. In: Encyclopedia of Social Network Analysis and Mining. 2nd Ed. Springer, 2018.

[HR21] Halawa, Hassan; Ripeanu, Matei: Position paper: bitemporal dynamic graph analytics. In: GRADES NDA. pp. 1–12, 2021.

[Io16] Iosup, Alexandru et al.: LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. Proc. of the VLDB Endow., 9(13):1317–1328, 2016.

[Iy21] Iyer, Anand Padmanabha; Pu, Qifan; Patel, Kishan; Gonzalez, Joseph E; Stoica, Ion: TEGRA: Efficient Ad-Hoc Analytics on Evolving Graphs. In: NSDI. pp. 337–355, 2021.

[JNB03] Jeong, Hawoong; Néda, Zoltan; Barabási, Albert-László: Measuring preferential attachment in evolving networks. EPL (Europhysics Letters), 61(4):567, 2003.

[KA12] Kim, Hyoungshick; Anderson, Ross: Temporal node centrality in complex networks. Physical Review E, 85(2):026107, 2012.

[Ko09] Kostakos, Vassilis: Temporal graphs. Physica A: Statistical Mechanics and its Applications, 388(6):1007–1023, 2009.

[Li15] Li, Yexin; Zheng, Yu; Zhang, Huichu; Chen, Lei: Traffic prediction in a bike-sharing system. In: Proc. of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems. pp. 1–10, 2015.

[LJ21] Lizardo, Omar; Jilbert, Isaac: , Graph Metrics. http://olizardo.bol.ucla.edu/classes/soc-111/lessons-winter-2022/4-lesson-graph-metrics.html, 2021. [Online; accessed 2022-10-01].

[Lo20] Long, Li; Abbas*, Khushnood; Ling*, Niu; Jafar Abbas, Syed: Ranking Nodes in Temporal Networks: Eigen Value and Node Degree Growth based. In: 2nd International Conference on Image Processing and Machine Vision. pp. 146–153, 2020.

[Ne01] Newman, Mark EJ: Clustering and preferential attachment in growing networks. Physical review E, 64(2):025102, 2001.

[Ni13] Nicosia, Vincenzo; Tang, John; Mascolo, Cecilia; Musolesi, Mirco; Russo, Giovanni; Latora, Vito: Graph metrics for temporal networks. In: Temporal Networks, pp. 15–40. Springer, 2013.

[RKC01] Riolo, Christopher S; Koopman, James S; Chick, Stephen E: Methods and measures for the description of epidemiologic contact networks. J Urban Health, 78(3):446–457, 2001.

[Ro21] Rost, Christopher; Gómez, Kevin; Fritzsche, Philip; Thor, Andreas; Rahm, Erhard: Exploration and Analysis of Temporal Property Graphs. In: EDBT. pp. 682–685, 2021.

[Ro22] Rost, Christopher; Gomez, Kevin; Täschner, Matthias; Fritzsche, Philip; Schons, Lucas; Christ, Lukas; Adameit, Timo; Junghanns, Martin; Rahm, Erhard: Distributed temporal graph analytics with GRADOOP. The VLDB Journal, 31:1–27, 2022.

[Sa20] Sahu, Siddhartha; Mhedhbi, Amine; Salihoglu, Semih; Lin, Jimmy; Özsu, M Tamer: The ubiquity of large graphs and surprising challenges of graph processing: extended survey. The VLDB Journal, 29(2):595–618, 2020.

[SE20] Smith, Keith M; Escudero, Javier: Normalised degree variance. Applied Network Science, 5(1):1–22, 2020.

[SK05] Saramäki, Jari; Kaski, Kimmo: Modelling development of epidemics with dynamic small-world networks. Journal of Theoretical Biology, 234(3):413–421, 2005.

[Sn81] Snijders, Tom AB: The degree variance: an index of graph heterogeneity. Social networks, 3(3):163–174, 1981.

[Ta72] Tarjan, Robert: Depth-first search and linear graph algorithms. SIAM Journal on Computing, 1(2):146–160, 1972.

[TBF17] Thompson, William Hedley; Brantefors, Per; Fransson, Peter: From static to temporal network theory: Applications to functional brain connectivity. Network Neuroscience, 1(2):69–99, 2017.

[Tl20] Tlebaldinova, Aizhan; Nugumanova, Aliya; Baiburin, Yerzhan; Zhantassova, Zheniskul; Karmenova, Markhaba; Ivanov, Andrey: Temporal Network Approach to Explore Bike Sharing Usage Patterns. In: VEHITS. pp. 129–136, 2020.

[Wa17] Wang, Zhiqiang; Pei, Xubin; Wang, Yanbo; Yao, Yiyang: Ranking the key nodes with temporal degree deviation centrality on complex networks. In: 2017 29th Chinese Control And Decision Conference (CCDC). IEEE, pp. 1484–1489, 2017.

[Wu14] Wu, Huanhuan; Cheng, James; Huang, Silu; Ke, Yiping; Lu, Yi; Xu, Yanyan: Path problems in temporal graphs. Proc. of the VLDB Endowment, 7(9):721–732, 2014.

[YvdHL17] Yao, Dong; van der Hoorn, Pim; Litvak, Nelly: Average nearest neighbor degrees in scale-free networks. arXiv preprint arXiv:1704.05707, 2017.

[Za16] Zaharia, Matei et al.: Apache Spark: A Unified Engine for Big Data Processing. Communications of the ACM, 59(11):56–65, 2016.