GraphTempo: An aggregation framework for evolving graphs

Evangelia Tsoukanara
University of Macedonia
Thessaloniki, Greece
etsoukanara@uom.edu.gr

Georgia Koloniari
University of Macedonia
Thessaloniki, Greece
gkoloniari@uom.edu.gr

Evaggelia Pitoura
University of Ioannina
Ioannina, Greece
pitoura@cse.uoi.gr

ABSTRACT

Graphs offer a generic abstraction for modeling entities and the interactions and relationships between them. Since most real-world graphs evolve over time, there is a need for models to explore the evolution of graphs over time. We introduce the GraphTempo model that allows aggregation both at the attribute and at the time dimension. We also propose an exploration strategy for navigating through the evolution of the graph based on identifying time intervals of significant growth, shrinkage or stability. This exploration strategy would be useful for example for identifying time periods of multiple collaborations between specific groups in a cooperation network, or of declining contacts between specific groups in a disease propagation network. We evaluate the performance and effectiveness of our strategy using two real graphs.

1 INTRODUCTION

Graphs offer an effective model for a variety of data by capturing the relationships and interactions between entities, ranging for example, from the co-authorship relation between people in a collaboration network, to the ratings between users and items in a recommendation system. Most such graphs are evolving as the real-world entities they represent and their relationships change through time. Thus, an interesting problem is to study the evolution of the graph, and summarize it so as to get insights and substantial information of the underlying data and its evolution.

There has been a lot of work on aggregation of static graphs e.g., [5, 26]. More recently, aggregation in temporal graphs has received some attention, with research focusing on temporal paths [6], extensions of the property model with time [19], conceptual modeling [13], graph analytics [16], visualization [23], and supporting different time granularities [1]. In this paper, we look into the novel problem of identifying important points in the evolution of the graph both in terms of stability but also of significant growth and shrinkage.

To this end, we introduce the *GraphTempo* framework for aggregating temporal graphs and exploring their evolution. We consider temporal attributed graphs where nodes have properties whose values may or not change with time. Our framework facilitates flexible aggregation both in the attribute and time dimensions.

Aggregation encompassing both attribute types and time allows us to study a network at a global level and detect patterns that are not identified when observing the graph at the individual vertex level. Further, aggregating network instances corresponding to different time intervals may reveal interesting patterns of evolution.

To motivate our work, consider a co-authorship network, where nodes represent authors and edges their collaborations. By aggregating authors based on one or more types of attributes for different time periods, we may discover interesting events which may be associated with external factors. For example, reporting an increase on the collaborations between female authors on a specific time period could be related with a diversity and inclusion action to empower female researchers and women in science. Considering the opposite angle, one may wish to assess the effect of such a diversity and inclusion action by studying the aggregated collaboration graph after and before the action. Aggregating authors by affiliation and nationality, in addition to time, may reveal biases that emerge through time with shrinkage between relationships of authors of different nationalities or on the other hand the growth of such relationships may be attributed to the effect of corresponding action plans that aim to improve international collaborations.

As another motivating scenario, consider the association of face-to-face proximity relations between individuals and the propagation of infectious diseases in the community. An empirical dataset with contacts between students and teachers, the time interval of their interaction, the class and the grade of the children is used in [12]. Since children usually spend more time in contact with children of the same class and the same grade, the authors show that a targeted closure strategy based on class and grade is efficient for the mitigation of influenza. Thus, temporal aggregation based on the individuals' attributes, contact duration and the homophily in the aggregated network may reveal the degree of risk of spreading an infectious disease to the rest of the community. Measuring shrinkage in contacts or disease spreading can be used to evaluate the effectiveness of applied mitigation measures, while detecting stable contacts indicates that further measures are required to counter the spread.

To support different levels of temporal resolution, we define temporal operators that aggregate the graphs that exist in two time intervals using either tight *intersection* semantics or more relaxed *union* semantics. We provide distinct and non-distinct attribute aggregation by grouping nodes based on their attribute values and the corresponding relationships between them, and present algorithms for the implementation of all temporal operators and aggregation types. We also show how aggregate graphs can be reused to compute other aggregate graphs more efficiently without having to access the original temporal graphs.

To model graph evolution, we define three types of events pertaining to the *growth*, *stability* and *shrinkage* of the graph. We define the *evolution graph* that captures the three types of evolution events by overlaying three different graphs defined using appropriate temporal operators for each event type. To explore graph evolution, we define the novel problem of finding pairs of intervals $(\mathcal{T}_1, \mathcal{T}_2)$ such that at least k events of interest have occurred between $\mathcal{T}_1$ and $\mathcal{T}_2$. Since the number of pairs of intervals to consider can grow exponentially, we propose an efficient exploration strategy that considers a fixed reference point and utilizes intersection and union semantics to extend intervals. We exploit monotonicity to further prune the exploration space.

© 2022 Copyright held by the owner/author(s). Published in Proceedings of the 25th International Conference on Extending Database Technology (EDBT), 29th March-1st April, 2022, ISBN 978-3-89318-086-8 on OpenProceedings.org.

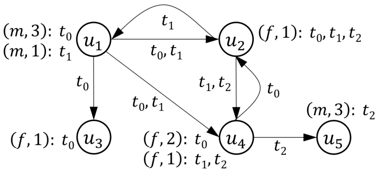


Figure 1: A temporal attributed graph.

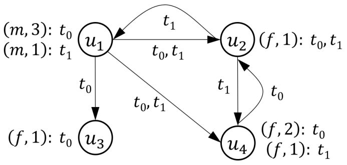


Figure 2: Union graph of the graph of Fig. 1 on $[t_0, t_1]$.

Finally, we present an experimental evaluation of our approach using two real datasets, aiming at both studying the efficiency of the proposed algorithms and the effectiveness of our proposed evolution exploration strategies.

The rest of the paper is structured as follows. In Section 2, we define the temporal operators, the attribute aggregation, and the evolution graph. In Section 3, we define our problem of exploring the graph evolution. In Section 4, we present our algorithms, and in Section 5 our experimental results. Section 6 summarizes related work and Section 7 offers conclusions.

2 THE GRAPHTEMPO MODEL

We assume an interval-based definition of a temporal graph. We use $\mathcal{T}$ to denote a set of intervals. Each temporal graph refers to such a set. We model the existences of edges and nodes by two timestamp functions τu and τe the domain of which is $\mathcal{T}$.

Definition 2.1 (Temporal Attributed Graph). A temporal attributed graph in $\mathcal{T}$ is a graph $G(V, E, \tau u, \tau e, A)$ where V is the set of nodes and E is the set of edges (u, v) with $u, v \in V$. Each node in V is associated with a timestamp $\tau u : V \longrightarrow \mathcal{T}$ where $\tau u(u)$ is the set of intervals during which u exists. Similarly each edge e in E is associated with a timestamp $\tau e : E \longrightarrow \mathcal{T}$ where $\tau e(e)$ is the set of intervals during which e exists. $A = \{A_1, A_2 \dots A_k\}$ is a set of k node attributes whose values may change over time, that is, for each $u \in V$ and $t \in \tau u(u)$, there is a k-dimensional tuple, $A(u, t) = \{A_1(u, t), A_2(u, t) \dots A_k(u, t)\}$, where $A_i(u, t)$ denotes the value of u at time $t \in \tau u(u)$ on the i-th attribute.

An attribute A_i is called *static*, if its value does not change with time, i.e., $A_i(u, t) = A_i(u, t')$, $\forall u \in V$ and $\forall t, t' \in \tau(u) \subseteq T$, and *time-varying* otherwise.

Figure 1 depicts an example of a temporal attributed graph for a collaboration graph in time period $\mathcal{T} = \{t_0, t_1, t_2\}$ consisting of, for simplicity, three distinct time points. The nodes have two attributes, "gender" a categorical attribute with values $\{m, f\}$ that is static and "#Publications" a numerical attribute with integer values that is time-varying.

Other temporal graph models.

Temporal graph models can be classified into: *duration-labeled*, where nodes have no properties and edges are labeled with a starting point and a duration; *interval-labeled*, where edges are timestamped with the time interval in which they are valid; and, *snapshot-based*, where a graph in an interval is given by a sequence of graph snapshots for each time point in the interval [6]. Our model follows the interval-labeled model. Several variations of the interval-labeled graph model have been defined, including TGraph [1] that associates graph entities with time points but treats adjacent points as intervals, Chronograph [2] that converts point to interval semantics, and the temporal property graph model (TPG) [3, 6, 19, 23]. Similarly to our model, the TPG model includes timestamps on nodes and attributes, and extends timestamps to set of intervals. TPG represents node objects, their attributes and values using different types of nodes. Instead, as in previous work in graph aggregation [5, 26], we do not represent attributes as different node types. Our concise representation allows us to clearly formulate the temporal operators and extend graph aggregation with time. Our approach can also be adapted for any graph representation that follows the interval-labeled temporal graph model.

2.1 Temporal Operators

We focus first on how to define temporal graphs on intervals of different lengths and combinations of intervals. To this end, we define a set of temporal operators that given a temporal attributed graph and some time intervals, define a new temporal graph defined on the combination of the given intervals.

Our first operator is *project* that defines a subgraph of the original temporal graph defined on specific time intervals.

Definition 2.2 (Time Project Operator). The projection G_1 of G in $\mathcal{T}_1 \subseteq \mathcal{T}$ is a temporal attributed graph at time interval $\mathcal{T}_1$ $G_1(V_1, E_1, \tau u_1, \tau e_1, A_1)$, where V_1 includes all nodes $u \in V$ for which $\mathcal{T}_1 \subseteq \tau u(u)$, E_1 all edges $e \in E$ for which $\mathcal{T}_1 \subseteq \tau e(e)$, and for all nodes $u \in V_1$, $\tau u_1(u) = \tau u(u) \cap \mathcal{T}_1$, and for all edges $e \in E_1$, $\tau e_1 = \tau e(e) \cap \mathcal{T}_1$ and A_1 includes the attributes for all nodes in V_1 for the interval $\mathcal{T}_1$.

Given two sets of time intervals, $\mathcal{T}_1$ and $\mathcal{T}_2$, the union operator defines a graph that includes the nodes and edges that exist in at least one time instant in $\mathcal{T}_1$ or $\mathcal{T}_2$.

Definition 2.3 (Union Operator). Given G and two sets of time intervals $\mathcal{T}_1$ and $\mathcal{T}_2$, the union graph is a temporal attributed graph $G_\cup(V_\cup, E_\cup, \tau u_\cup, \tau e_\cup, A_\cup)$ defined in $(\mathcal{T}_1, \mathcal{T}_2)$, where $V_\cup = \{u | (\tau u(u) \cap \mathcal{T}_1 \neq \emptyset)\ or\ (\tau u(u) \cap \mathcal{T}_2 \neq \emptyset)\}$, $E_\cup = \{e | (\tau e(e) \cap \mathcal{T}_1 \neq \emptyset)\ or\ (\tau e(e) \cap \mathcal{T}_2 \neq \emptyset)\}$, for all nodes $u \in V_\cup$, $\tau u_\cup(u) = \tau u(u) \cap (\mathcal{T}_1 \cup \mathcal{T}_2)$, for all edges $e \in E_\cup$, $\tau e_\cup(e) = \tau e(e) \cap (\mathcal{T}_1 \cup \mathcal{T}_2)$ and $A_\cup$ includes the attributes for all nodes $u \in V_\cup, \forall t \in \tau u_\cup(u)$.

Figure 2 depicts the union graph $G_\cup$ of the temporal graph of Fig. 1 on (t_0, t_1).

The intersection operator captures the stable part of the graph, i.e., the set of edges and nodes that exist in both intervals.

Definition 2.4 (Intersection Operator). Given G and two sets of time intervals $\mathcal{T}_1$ and $\mathcal{T}_2$, the intersection graph is a temporal attributed graph $G_\cap(V_\cap, E_\cap, \tau u_\cap, \tau e_\cap, A_\cap)$ defined in $(\mathcal{T}_1, \mathcal{T}_2)$, where $V_\cap = \{u | (\tau u(u) \cap \mathcal{T}_1 \neq \emptyset)\ and\ (\tau u(u) \cap \mathcal{T}_2 \neq \emptyset)\}$, $E_\cap = \{e | (\tau e(e) \cap \mathcal{T}_1 \neq \emptyset)\ and\ (\tau e(e) \cap \mathcal{T}_2 \neq \emptyset)\}$, for all nodes $u \in V_\cap$, $\tau u_\cap(u) = \tau u(u) \cap (\mathcal{T}_1 \cup \mathcal{T}_2)$, for all edges $e \in E_\cap$, $\tau e_\cap = \tau e(e) \cap (\mathcal{T}_1 \cup \mathcal{T}_2)$ and $A_\cap$ includes the attributes for all nodes $u \in V_\cap, \forall t \in \tau u_\cap(u)$.

The difference operator $\mathcal{T}_1 - \mathcal{T}_2$, given that $\mathcal{T}_1$ precedes $\mathcal{T}_2$, captures the part of the graph that existed in $\mathcal{T}_1$ but not in $\mathcal{T}_2$, that is the nodes and edges that were deleted.

Definition 2.5 (Difference Operator). Given G and two sets of time intervals $\mathcal{T}_1$ and $\mathcal{T}_2$, the difference graph between $\mathcal{T}_1$ and $\mathcal{T}_2$

is a temporal attributed graph $G_-(V_-, E_-, \tau u_-, \tau e_- A_-)$ defined in $\mathcal{T}_1 \cap \mathcal{T}_2$, where $V_- = \{u | (\tau u(u) \cap \mathcal{T}_1 \neq \emptyset)$ *and* $((\tau u(u) \cap \mathcal{T}_2 = \emptyset)$ *or* $(\exists (u,v) \in E_-)\}$, $E_- = \{u | (\tau e(e) \cap \mathcal{T}_1 \neq \emptyset)$ *and* $(\tau e(e) \cap \mathcal{T}_2 = \emptyset)\}$, for all nodes $u \in V_-$, $\tau u_-(u) = \tau u(u) \cap \mathcal{T}_1$, for all edges $e \in E_-$, $\tau e_- = \tau e(e) \cap \mathcal{T}_1$ and A_- includes the attributes for all nodes $u \in V_-, \forall t \in \tau u_-(u) \cup \tau e_-(u,v)$.

The difference operator is not symmetric. The difference $\mathcal{T}_2 - \mathcal{T}_1$, given that $\mathcal{T}_1$ precedes $\mathcal{T}_2$, captures the part of the graph that exists in $\mathcal{T}_2$ but not in $\mathcal{T}_1$, that is the new nodes and edges.

2.2 Graph Aggregation

We might describe graph aggregation as the operation of grouping nodes together based on (some of) their attributes, while taking network structure into consideration. So, each aggregate node (node in the aggregate graph) corresponds to a set of the original nodes and aggregate edges are built based on the interactions of the aggregate nodes in the original network.

Definition 2.6 (Graph Aggregation). Given a temporal attributed graph $G(V, E, \tau u, \tau e, A)$, in $\mathcal{T}$, and n aggregate attributes, $1 \leq n \leq k$, $A'_1, A'_2, .., A'_n \in A$, the aggregated graph G' is defined as a weighted graph $G'(V', E', W_{V'}, W_{E'}, A')$, where for each distinct tuple of $A'_1(u,t), A'_2(u,t), \ldots, A'_n(u,t)$, $u \in V$ and $t \in \tau u(u)$, there is a node $u' \in V'$ with $A'_i(u') = A_i(u,t), \forall i, 1 \leq i \leq n$, and there is an edge $e' \in E'$ between $u', v' \in V'$, if and only if $\exists e \in E$ between nodes $u, v \in V$ with $A'_i(u') = A_i(u,t)$ and $A'_i(v') = A_i(v,t), \forall i, 1 \leq i \leq n$. The weight $w_{V'}$ is an aggregate function f_V defined upon vertices and the weight w'_e of e' is an aggregate function f_E defined upon edges.

We use COUNT as our aggregation function for both node and edges, in our model and the rest of this paper. However other aggregations may be supported, if edges are attributed as well.

We discern between two types of aggregation, *distinct* aggregation (denoted as DIST) and *non-distinct* aggregation (denoted as ALL). Distinct aggregation is about aggregating on unique nodes and edges of the original graph, while, in non-distinct aggregation, duplicates are not identified and counted each time they appear. Thus, using COUNT as our aggregate function, weights are calculated in the two cases as follows.

- For $G'_{DIST}(V', E', W_{V'}, W_{E'}, A')$, the weight $w_{V'}(u')$ of the attribute tuple $A'_1(u'), A'_2(u'), \ldots, A'_n(u')$ is defined as the number of distinct $u \in V$ with $A'_i(u') = A_i(u,t), 1 \leq i \leq n$ at any time point $t \in T$.
- For $G'_{ALL}(V', E', W_{V'}, W_{E'}, A')$, the weight $w_{V'}(u')$ of the attribute tuple $A'_1(u'), A'_2(u'), \ldots, A'_n(u')$ is defined as the $\sum_u |t_u|$, for all $u \in V$, where $|t_u|$ denotes the number of times points $t \in T$ that with $A'_i(u') = A_i(u,t), 1 \leq i \leq n$.

In G'_{DIST}, we count all appearances of an attribute tuple on the same node as one, while in G'_{ALL}, each tuple is counted each time it appears either on the same or different nodes.

Figure 3(a-c) depicts the aggregate graphs on gender and publications of the projection graphs of t_0, t_1 and t_2 respectively of the graph of Fig. 1. As we consider aggregate graphs on a time point instead of interval, there is no difference between G'_{DIST} and G'_{ALL}. We can see the difference in Fig. 3(d-e) that shows, for the union graph of Fig. 2, G'_{DIST} (Fig. 3d) and G'_{ALL} (Fig. 3e) respectively. The weight for the node *'f,1'* in G'_{DIST} is equal to 3, as 3 distinct nodes appear with this attribute contribution in the union graph, while in G'_{ALL} it is equal to 4 as there are 4 total appearances of this combination in the union graph.

2.3 Evolution Graph

In addition to the temporal operators that are based on set operations and graph aggregation, since we want to capture graph evolution, i.e., the changes that occur in a graph through time, we define the *evolution graph.*

Definition 2.7 (Evolution Graph). Given G and two sets of time intervals $\mathcal{T}_1$ and $\mathcal{T}_2$, the evolution graph between $\mathcal{T}_1$ and $\mathcal{T}_2$ is a temporal attributed graph $G_>(V_>, E_>, \tau u_>, \tau e_> A_>)$ in $(\mathcal{T}_1, \mathcal{T}_2)$, with $G_\cap$ the intersection graph defined in $(\mathcal{T}_1, \mathcal{T}_2)$, G_- the difference graph between $\mathcal{T}_1$ and $\mathcal{T}_2$ and G'_- the difference graph between $\mathcal{T}_2$ and $\mathcal{T}_1$, where $V_> = V_\cap \cup V_- \cup V'_-$, $E_> = E_\cap \cup E_- \cup E'_-$, for nodes $u \in V_\cap$, $\tau u_>(u) = \tau u_\cap(u)$, for nodes $u' \in V_-$, $\tau u_>(u') = \tau u_-(u')$, for nodes $u'' \in V'_-$, $\tau u_>(u'') = \tau u'_-(u'')$, for edges $e \in E_\cap$, $\tau e_>(e) = \tau e_\cap(e)$, for edges $e' \in E_-$, $\tau e_>(e') = \tau e_-(e')$, for edges $e'' \in E'_-$, $\tau e_>(e'') = \tau e'_-(e'')$, and $A_>$ includes the attributes for all nodes $u \in V_>, \forall t \in \tau u_>(u)$

An evolution graph is therefore defined as a combination of three other graphs, one defined based on the intersection and two on the difference operator. Thus, we may discern between different types of nodes and edges. The evolution graph consists of nodes and edges that have stayed in the graph in both intervals, others that disappear in the later interval and others that appear in the second interval while not in the first, capturing therefore in detail the evolution of the graph. In particular, for an evolution graph $G_>$ defined on $\mathcal{T}_1, \mathcal{T}_2$:

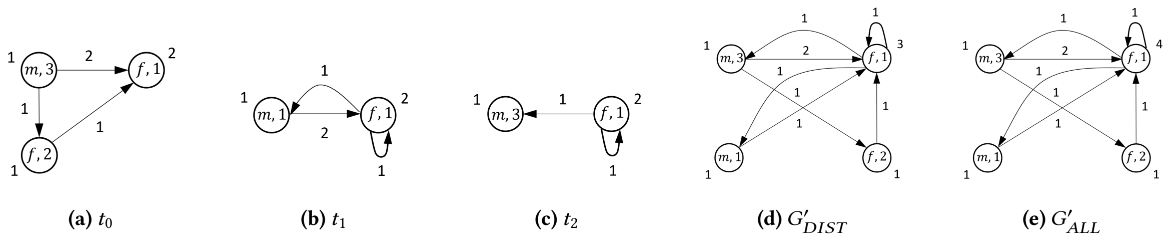
- Nodes $u \in V_>$ for which $\tau u_>(u) \cap \mathcal{T}_1 \neq \emptyset$ and $\tau u_>(u) \cap \mathcal{T}_2 \neq \emptyset$ are nodes that remain during the evolution of the graph from $\mathcal{T}_1$ to $\mathcal{T}_2$, i.e., these are the nodes belonging to $V_\cap$ and capture *stability.*
- Nodes $u' \in V_>$ for which $\tau u_>(u') \cap \mathcal{T}_1 \neq \emptyset$ and $\tau u_>(u') \cap \mathcal{T}_2 = \emptyset$ are nodes that exist in the graph in interval $\mathcal{T}_1$ but disappear at interval $\mathcal{T}_2$, i.e., these are the nodes belonging to V_- defined on $\mathcal{T}_1 - \mathcal{T}_2$ and capture *shrinkage.*
- Nodes $u'' \in V_>$ for which $\tau u_>(u'') \cap \mathcal{T}_1 = \emptyset$ and $\tau u_>(u'') \cap \mathcal{T}_2 \neq \emptyset$ are nodes that did not exist in the graph in interval $\mathcal{T}_1$ but have appeared at interval $\mathcal{T}_2$, i.e., these are the nodes belonging to V_- defined on $\mathcal{T}_2 - \mathcal{T}_1$ and capture *growth.*

Similarly, we may discern between the edges. Figure 4a shows the evolution graph of the graph of Fig.1 from time point t_0 to t_1, where labels *St*, *Gr* and *Shr* are used to differentiate between entities with stability, growth and shrinkage respectively.

As an evolution graph is derived by the combination of an intersection and two difference graphs, aggregation can be applied similarly by considering each such graph separately or as a whole. In the latter case, the three aggregate graphs are overlaid and different weights corresponding to each entity are used so as to discern between growth, shrinkage and stability. Figure 4b depicts the aggregation of the evolution graph of Fig. 4a, where we can discern the weights for stability, growth and shrinkage. For instance, node $(f, 1)$, corresponding to female authors with 1 publication, has: a) stability weight 1, as it had 1 stable appearance on node u_2 at both t_0 and t_1, b) growth weight 1, because a new appearance on u_4 occurs at t_1, and c) shrinkage weight 1, because one appearance on u_3 at t_0 is removed in t_1.

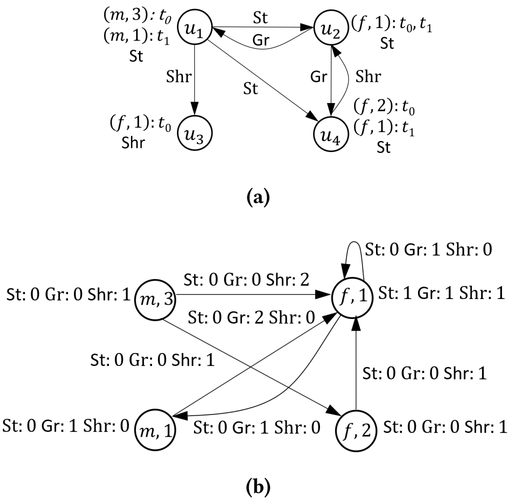
3 EXPLORATION

Our goal is to use the idea of the evolution graph so as to detect interesting events in the behavior of the graph. Based on the different types of nodes and edges in the graph, we discern three



(a) t_0 **(b)** t_1 **(c)** t_2 **(d)** G'_{DIST} **(e)** G'_{ALL}

Figure 3: Aggregate graphs on (Gender, #Publications) for (a-c) the graph of Fig.1 per time point and (d-e) the graph of Fig. 2.



(a)

(b)

Figure 4: (a) Evolution graph for t_0, t_1 and (b) its aggregation on (Gender, #Publications) for the graph of Fig. 1.

types of events: (i) *growth*, which concerns the addition of entities, nodes or edges, in the graph, (ii) *shrinkage*, which concerns the deletion of entities from the graph, and (iii) *stability*, which concerns entities that remain stable in the graph.

We want to study the evolution of a graph in time intervals of different length, for example, find events that occur between two months, six months or two years, etc. We use a threshold based approach and considering a specific type of entities, we define our problem as: *detect pairs of intervals between which at least k events have occurred.*

3.1 Union and Intersection Semantics

We are interested on changes on aggregate graphs, and as the number of aggregate graphs defined on different intervals one has to study is exponential, we need a way to efficiently explore the corresponding search space. Based on our problem definition, we are interested in considering time intervals of various lengths and not sets of intervals, thus, we require combining successive intervals to derive intervals of greater length.

Let $\mathcal{T}_i, 1 \leq i \leq n$, be the intervals with shortest length. If we consider $\mathcal{T}_i$ as the elements of a set $\mathcal{T}$, we may construct the lattice of the powerset $\mathcal{T}$. Since we want to extend intervals based on combining successive intervals, we only focus on a sub lattice of the powerset lattice of $\mathcal{T}$, that is the tree that combines at each level t of the lattice two elements of the previous level $t-1$ if they differ only by one element. As a powerset lattice is defined on union and intersect set operators, it follows by the definitions of the temporal operators union and intersection, that the graphs defined on the time intervals of the sub lattice also form a sub lattice with respect to these operations.

Therefore, we can define a temporal graph using either *union* semantics by considering the respective union semi-lattice, or *intersection* semantics by considering the respective intersection semi-lattice. The first approach adopts a more relaxed view, where the graph defined on a union of time intervals contains all entities that exist in any of the two original time intervals. On the other hand, using intersection semantics, we adopt a stricter view, in which the intersection graph contains only entities that exist in both the original time intervals.

Definition 3.1 (Monotonically Increasing). We define aggregation as monotonically increasing or increasing w.r.t. a temporal operator $(\cdot)$, if given intervals $\mathcal{T}_k, \mathcal{T}_i, \mathcal{T}_j$ such that $\mathcal{T}_i \subseteq \mathcal{T}_j$ and aggregate graphs $G_i(V_i, E_i, W_{V_i}, W_{E_i}, A)$ and $G_j(V_j, E_j, W_{V_j}, W_{E_j}), A$ defined on $\mathcal{T}_k \cdot \mathcal{T}_i$ and $\mathcal{T}_k \cdot \mathcal{T}_j$ respectively, it holds that (i) $\forall v \in V_j$, if $v \in V_i$ then $w_{V_i}(v) \leq w_{V_j}(v)$, and (ii) $\forall e \in E_j$, if $e \in E_i$ then $w_{E_i}(e) \leq w_{E_j}(e)$.

Definition 3.2 (Monotonically Decreasing). We define aggregation as monotonically decreasing or decreasing w.r.t. a temporal operator $(\cdot)$, if given intervals $\mathcal{T}_k, \mathcal{T}_i, \mathcal{T}_j$ such that $\mathcal{T}_i \subseteq \mathcal{T}_j$, and aggregate graphs $G_i(V_i, E_i, W_{V_i}, W_{E_i}, A)$, $G_j(V_j, E_j, W_{V_j}, W_{E_j}, A)$ defined on $\mathcal{T}_k \cdot \mathcal{T}_i$ and $\mathcal{T}_k \cdot \mathcal{T}_j$ respectively, it holds that (i) $\forall v \in V_j$, if $v \in V_i$ then $w_{V_i}(v) \geq w_{V_j}(v)$, and (ii) $\forall e \in E_j$, if $e \in E_i$ then $w_{E_i}(e) \geq w_{E_j}(e)$.

Regarding union and intersection semantics it easily follows from their definition that:

Lemma 3.3. *Aggregation is monotonically increasing with respect to union and monotonically decreasing with respect to intersection.*

As union is monotonically increasing, it follows that expanding intervals using union will increase the size of the aggregate graphs and the number of events we find. Thus, the highest number of events of interest would be detected when considering the largest intervals. Consequently, in this case it is interesting to detect the minimal interval pairs at which at least k events occur.

Definition 3.4 (Minimal Interval Pair). A minimal interval pair $\mathcal{T}_i, \mathcal{T}_j$ for a given threshold k is defined as a pair of intervals for which at least k events occurred from $\mathcal{T}_i$ to $\mathcal{T}_j$, if $\mathcal{T}_i$ precedes $\mathcal{T}_j$, or from $\mathcal{T}_j$ to $\mathcal{T}_i$ if $\mathcal{T}_j$ precedes $\mathcal{T}_i$ and for the same $\mathcal{T}_i$, $\nexists \mathcal{T}_{j'} \subset \mathcal{T}_j$ such that at least k events occurred from $\mathcal{T}_i$ to $\mathcal{T}_{j'}$ or $\mathcal{T}_{j'}$ to $\mathcal{T}_i$ respectively.

On the other hand, as intersection is monotonically decreasing, it follows that expanding intervals using intersection will reduce the size of the aggregate graphs and the number of events we find. Therefore, the highest number of events of interest would be detected when considering the smallest intervals. Consequently, in this case it is interesting to detect the maximal interval pairs at which at least k events occur.

Definition 3.5 (Maximal Interval Pair). A maximal interval pair $\mathcal{T}_i, \mathcal{T}_j$ for a given threshold k is defined as a pair of intervals for which at least k events occurred from $\mathcal{T}_i$ to $\mathcal{T}_j$, if $\mathcal{T}_i$ precedes $\mathcal{T}_j$, or from $\mathcal{T}_j$ to $\mathcal{T}_i$ if $\mathcal{T}_j$ precedes $\mathcal{T}_i$, and for the same $\mathcal{T}_i$, $\nexists \mathcal{T}_{j'} : \mathcal{T}_j \subset \mathcal{T}_{j'}$ for which at least k events occurred from $\mathcal{T}_i$ to $\mathcal{T}_{j'}$, or $\mathcal{T}_{j'}$ to $\mathcal{T}_i$ respectively.

Based on the above, we define our problem as follows.

Definition 3.6 (Problem Definition). Given an attributed graph G and a time period $\mathcal{T}$ and a user-defined threshold k, find the minimal and maximal interval pairs in which at least k events of either stability, growth or shrinkage occur.

3.2 Stability

Let us focus on the stability issue first. To determine the minimal or maximal interval pairs on which at least k entities remain stable, we need to consider all possible pairs of intervals using union or intersection semantics respectively. We propose a more efficient exploration of the candidate interval pairs by exploiting their monotonicity properties.

Therefore, given two time intervals $\mathcal{T}_{old}$ and $\mathcal{T}_{new}$, where $\mathcal{T}_{old}$ precedes $\mathcal{T}_{new}$, we study stability by studying the aggregate graph $G_\cap$ defined on $\mathcal{T}_{old}$, $\mathcal{T}_{new}$. We propose keeping one of the two ends of the interval fixed, as a time point of reference, and gradually extending the other end of the interval by exploiting the union or the intersection semi-lattice respectively. When our focus is on the original graph then we maintain $\mathcal{T}_{old}$ fixed as our reference point, while when our focus is on the latest state of the graph, $\mathcal{T}_{new}$ is viewed as our fixed reference point.

We denote as $result(G)$ the number of events of interest that belong in the aggregate graph. We first consider union semantics, thus, our problem is defined as finding the minimal intervals pairs in which at least k entities remain stable.

Without loss of generality, we maintain $\mathcal{T}_{old}$ and gradually extend $\mathcal{T}_{new}$. Given $\mathcal{T}_i$, $1 \leq i \leq n$, we apply the following steps:

(1) First compute the aggregate graphs for all pairs $\mathcal{T}_i \cap \mathcal{T}_{i+1}$, $1 \leq i < n$.
(2) If for any aggregate graph G defined on $\mathcal{T}_i \cap \mathcal{T}_{i+1}$, it holds that $result(G) \geq k$, return G on $\mathcal{T}_i \cap \mathcal{T}_{i+1}$ and prune any pair with $\mathcal{T}_i$ as its left end,
(3) For each pair $\mathcal{T}_i \cap \mathcal{T}_{i+1}$ that continues, extend its right end by substituting $\mathcal{T}_{i+1}$ with its right child in the union semi-lattice (i.e., $\mathcal{T}_{i+1} \cup \mathcal{T}_{i+2}$, and apply step 2.
(4) The previous is repeated by extending the right end of each surviving interval with its right child in the union semi-lattice (i.e., $\mathcal{T}_{i+1} \cup \mathcal{T}_{i+2} \cup \cdots \cup \mathcal{T}_{i+k}$) until either $i+k = n$ or no interval survives further.

We refer to this algorithm as *Union Exploration, (U-Explore)*. U-Explore can be adapted to perform extension on the left end of each interval, i.e., by maintaining $\mathcal{T}_{new}$ fixed, and extending $\mathcal{T}_{old}$. The difference is that $\mathcal{T}_{old}$ is substituted in the next step by its left child in the union semi-lattice.

Theorem 3.7. *The minimal interval pairs for stability derived by extending $\mathcal{T}_{new}$ are not equal to those derived by extending $\mathcal{T}_{old}$.*

Proof. Let us consider of a graph consisting of n (> 2) time points. $T_i \cap T_{i+1}$ forms the first of our initial pairs. T_i representing T_{old} cannot be extended as there is no preceding time point. Therefore, w.r.t. T_i as point of reference, $T_i \cap T_{i+1}$ is the only available pair. T_{new}, that is T_{i+1} will be extended to $T_{i+1} \cup T_{i+2}, \ldots, T_{i+1} \cup T_{i+2} \cup \cdots \cup T_n$, concluding to different pairs of intervals for the same point of reference. □

Let us now focus on intersection semantics, thus, our problem is defined as finding the maximal intervals pairs in which at least k entities (nodes or relationships) remain stable. Without loss of generality, let as assume that we maintain $\mathcal{T}_{old}$ and gradually extend $\mathcal{T}_{new}$. Given $\mathcal{T}_i$, $1 \leq i \leq n$, we apply the following steps:

(1) First compute the aggregate graphs for all pairs $\mathcal{T}_i \cap \mathcal{T}_{i+1}$, $1 \leq i < n$.
(2) If for any aggregate graph G defined on $\mathcal{T}_i \cap \mathcal{T}_{i+1}$, it holds that $result(G) \geq k$, add G to candidate set C, else prune any pair with $\mathcal{T}_i$ as its left end.
(3) For each pair $\mathcal{T}_i \cap \mathcal{T}_{i+1}$ that continues, extend its right end by substituting $\mathcal{T}_{i+1}$ with its right child in the intersection semi-lattice (i.e., $\mathcal{T}_{i+1} \cap \mathcal{T}_{i+2}$) and apply step 2, and for any graph G added to C, remove from C its predecessor (i.e., the aggregate graph defined on $\mathcal{T}_i \cap \mathcal{T}_{i+1}$).
(4) The previous is repeated by extending the right end of each surviving interval with its right child in the intersection semi-lattice (i.e., $\mathcal{T}_{i+1} \cup \mathcal{T}_{i+2} \cup \cdots \cup \mathcal{T}_{i+k}$) until either $i+k = n$ or no interval survives further.
(5) The final candidate set C is returned.

We refer to this algorithm as *Intersection Exploration, (I-Explore)*. I-Explore can be adapted to perform extension on the left end of each interval, i.e., by maintaining $\mathcal{T}_{new}$ fixed, and extending $\mathcal{T}_{old}$. The difference is that $\mathcal{T}_{old}$ is substituted in the next step by its left child in the intersection semi-lattice.

Theorem 3.8. *The maximal interval pairs for stability derived by extending $\mathcal{T}_{new}$ are equivalent to those derived by extending $\mathcal{T}_{old}$.*

Proof. Given that $T_i \cap (T_{i+1} \cap T_{i+2}) = (T_i \cap T_{i+1}) \cap T_{i+2}$ with points of reference T_i and T_{i+2} respectively, we derive that pairs of intervals with different points of reference give the same results if the time points participating in the pairs of intervals are identical. Therefore, extending T_{new} for all initial pairs concludes to the same maximal pairs of intervals when extending T_{old}. □

3.3 Growth

Given two time intervals $\mathcal{T}_{old}$ and $\mathcal{T}_{new}$, where $\mathcal{T}_{old}$ precedes $\mathcal{T}_{new}$, we study growth by studying the aggregate graph of $\mathcal{T}_{new} - \mathcal{T}_{old}$, i.e., the newer graph minus the old graph to evaluate the new additions.

Let us first consider union semantics, thus, our problem is defined as finding the minimal intervals pairs in which at least k entities (nodes or relationships) are added.

As difference is not a symmetric operator, our results differ when extending $\mathcal{T}_{old}$ in contrast to $\mathcal{T}_{new}$. When extending $\mathcal{T}_{old}$ we expect the $result(G)$ to decrease, while extending $\mathcal{T}_{new}$ we expect causes an increase to $result(G)$.

Let us first extend $\mathcal{T}_{old}$. We adopt U-Explore by computing at step (1), instead of the intersection, the difference graphs $\mathcal{T}_{i+1} - \mathcal{T}_i$, and then apply step (2). We claim that we do not need to proceed to further steps as for any aggregate graph for which $result(G) < k$ extending the right interval will not increase the result size on the derived aggregate graph.

Lemma 3.9. *The difference $\mathcal{T}_{new} - \mathcal{T}_{old}$ is monotonically decreasing when we extend $\mathcal{T}_{old}$ and monotonically increasing when we extend $\mathcal{T}_{new}$, using union semantics.*

Table 1: Exploration Intervals Properties

Event	Type	Case	Left	Right	Mon. Increasing	Mon. Decreasing	⊆ of
Growth	Min.	$\mathcal{T}_{new}$ - $\mathcal{T}_{old}(\cup)$	t.p.	t.p.		✓	$\mathcal{T}_{new}(\cup)$ - $\mathcal{T}_{old}$
		$\mathcal{T}_{new}(\cup)$ - $\mathcal{T}_{old}$	t.p. / interval	t.p.	✓		
	Max.	$\mathcal{T}_{new}$ - $\mathcal{T}_{old}(\cap)$	t.p.	longest interval	✓		
		$\mathcal{T}_{new}(\cap)$ - $\mathcal{T}_{old}$	t.p. / interval	t.p.		✓	
Shrinkage	Min.	$\mathcal{T}_{old}(\cup)$ - $\mathcal{T}_{new}$	t.p. / interval	t.p.	✓		
		$\mathcal{T}_{old}$ - $\mathcal{T}_{new}(\cup)$	t.p.	t.p.		✓	$\mathcal{T}_{old}(\cup)$ - $\mathcal{T}_{new}$
	Max.	$\mathcal{T}_{old}(\cap)$ - $\mathcal{T}_{new}$	t.p. / interval	t.p.		✓	
		$\mathcal{T}_{old}$ - $\mathcal{T}_{new}(\cap)$	t.p.	longest interval	✓		
Stability	Min.	$\mathcal{T}_{old}(\cup) \cap \mathcal{T}_{new}$	t.p. / interval	t.p.	✓		
		$\mathcal{T}_{new}(\cup) \cap \mathcal{T}_{old}$	t.p. / interval	t.p.	✓		
	Max.	$\mathcal{T}_{old}(\cap) \cap \mathcal{T}_{new}$	t.p. / interval	t.p.		✓	$\mathcal{T}_{new}(\cap) \cap \mathcal{T}_{old}$
		$\mathcal{T}_{new}(\cap) \cap \mathcal{T}_{old}$	t.p. / interval	t.p.		✓	$\mathcal{T}_{old}(\cap) \cap \mathcal{T}_{new}$

Proof. Let $result(G)$ the result weights of aggregate graph G on $\mathcal{T}_{new} - \mathcal{T}_{old}$. Since union is monotonically increasing, for the aggregate graph on $\mathcal{T}_{old}$, $result(G_{old})$ is at most equal to the weights $result(G_{\cup old})$ for the aggregate graph defined on the union based extension of $\mathcal{T}_{old}$. As $result(G_{new})$ for the aggregate graph defined on $\mathcal{T}_{new}$ remains the same, it follows that for $result(G')$ of the aggregate graph on $\mathcal{T}_{new} - \mathcal{T}_{\cup old}$ it holds that $result(G) \leq result(G')$. Similarly, by extending $\mathcal{T}_{new}$ to $\mathcal{T}_{\cup new}$, the result of $T_{\cup new} - T_{old}$ is at least equal to $result(G)$. □

Let us now consider extending $\mathcal{T}_{new}$. Again, we adopt U-Explore, by computing the difference aggregate graphs $\mathcal{T}_{i+1} - \mathcal{T}_i$ instead of the intersection aggregate graphs at the respective steps. The rest of the steps are applied as defined.

Next, we focus on intersection semantics and determine the maximal intervals pairs in which at least k entities are added. We first extend $\mathcal{T}_{old}$ using intersection semantics.

Lemma 3.10. *$\mathcal{T}_{new} - \mathcal{T}_{old}$ is monotonically increasing when we extend $\mathcal{T}_{old}$ and monotonically decreasing when we extend $\mathcal{T}_{new}$, using intersection semantics.*

Proof. Let $result(G)$ the result weights of aggregate graph G on $\mathcal{T}_{new} - \mathcal{T}_{old}$. As intersection is monotonically decreasing, for the aggregate graph on $\mathcal{T}_{old}$, $result(G_{old})$ is at least equal to the weights $result(G_{\cap old})$ for the aggregate graph defined on the intersection based extension of $\mathcal{T}_{old}$. As $result(G_{new})$ for the aggregate graph defined on $\mathcal{T}_{new}$ remains the same, it follows that for $result(G')$ of the aggregate graph on $\mathcal{T}_{new} - \mathcal{T}_{\cap old}$ it holds that $result(G) \geq result(G')$. Similarly, by extending $\mathcal{T}_{new}$ to $\mathcal{T}_{\cap new}$, the result of $T_{\cap new} - T_{old}$ is at most equal to $result(G)$. □

Based on the above lemma, the maximal interval pairs are: each point of reference $\mathcal{T}_{new}$, with the longest possible $\mathcal{T}_{old}$ as long as for the defined aggregate graph, $result(G) \geq k$.

When extending $\mathcal{T}_{new}$, I-Explore is adopted appropriately. In all respective steps, we compute the aggregate difference graphs $\mathcal{T}_{i+1} - \mathcal{T}_i$, and then apply the rest of the steps accordingly.

As difference is not a symmetrical operator to exploit monotonicity properties either $\mathcal{T}_{new}$ or $\mathcal{T}_{old}$ should be extended. When we extend both $\mathcal{T}_{new}$ and $\mathcal{T}_{old}$, difference is non-monotonous irrespectively to the semantics (union or intersection) used.

3.4 Shrinkage

Given two time intervals $\mathcal{T}_{old}$ and $\mathcal{T}_{new}$, where $\mathcal{T}_{old}$ precedes $\mathcal{T}_{new}$, we study shrinkage based on the aggregate graph of $\mathcal{T}_{old} - \mathcal{T}_{new}$, i.e., the old graph minus the new one to evaluate deletions.

With union semantics, our problem is defined as finding the minimal intervals pairs in which at least k entities (nodes or relationships) are deleted. When extending $\mathcal{T}_{old}$, we adapt U-Explore by computing the difference graphs $\mathcal{T}_i - \mathcal{T}_{i+1}$. On the other hand, when extending $\mathcal{T}_{new}$, as according to lemma 3.9, the operator is monotonically decreasing, we only need to evaluate the difference aggregate graphs on pairs of the shortest intervals (i.e., step 1 of U-Explore) and check their result with respect to k.

With intersection semantics we determine the maximal intervals pairs in which at least k entities are deleted. When extending $\mathcal{T}_{new}$, I-Explore is adapted appropriately by computing the aggregate difference graphs $\mathcal{T}_i - \mathcal{T}_{i+1}$. When extending $\mathcal{T}_{old}$, according to lemma 3.10 the operator is monotonically increasing and again for each reference point $\mathcal{T}_{new}$, the longest possible $\mathcal{T}_{old}$ forms the maximal interval and it suffices to check whether $result(G) \geq k$.

All cases are summarized on Table 1. Column *Case* shows which is the reference point and which interval is extended using either union (∪) or intersection (∩) semantics. Columns *Left* and *Right* refer to the left and right interval in the result interval pairs. One of the elements in the pair is always a time point (t.p.) while the other can be either of type time point (t.p.) or interval and longest interval. The last column shows the relationships between the results of different cases, and in particular when the result of one case is a subset of the result of another case.

3.5 Initialization of k

The appropriate value of the threshold k depends each time on the given graph and its data. To configure this threshold, we propose an initial value for k, indicating a threshold of interestingness and acting as a starting point so as to attain some results and then continue with its tuning. In particular, given the type of entity we are interested in, we initialize k to a value w_{th} which is defined as the minimum or the maximum weight of the given type of entity in the aggregation graph for any graph defined on the temporal aggregation of pairs of consecutive time points. For stability, we compute the aggregate graphs on intersections of pairs of consecutive time points, while for growth and shrinkage we compute the aggregate graphs on the appropriate difference graphs. Further, for a monotonically increasing operator, we start with w_{th} as the minimum aggregation weight for a given entity

Table 2: Arrays V, S for Gender, and A for #Publications for the graph of Fig. 1.

Id	t_0	t_1	t_2
u_1	1	1	0
u_2	1	1	1
u_3	1	0	0
u_4	1	1	1
u_5	0	0	1

Id	
u_1	m
u_2	f
u_3	f
u_4	f
u_5	m

Id	t_0	t_1	t_2
u_1	3	1	-
u_2	1	1	1
u_3	1	-	-
u_4	2	1	1
u_5	-	-	3

type, and increase it gradually, while for the monotonically decreasing operators w_{th} corresponds to the maximum aggregation weight and is gradually decreased as the exploration progresses.

4 ALGORITHMS

To store an attributed temporal graph $G(V, E, \tau u, \tau e, A)$ defined in a set of time intervals $\mathcal{T}$, we maintain separate structures for V, E and A. For each $v \in V$, $\tau u(u)$ is represented by a binary vector of size $|\mathcal{T}|$, where each element in the vector corresponds to a time $t \in \mathcal{T}$, and is 1 iff $t \in \tau u(v)$. Combining the temporal information for all nodes, we store these vectors as a labeled array $\mathbf{V}$ with $|V|$ rows labeled with the nodes ids and $|\mathcal{T}|$ columns labeled with the time t. A similar representation is deployed for the edges that are also stored in a labeled array $\mathbf{E}$, with $|E|$ rows labeled with the edges ids denoted as the edges end points (u, v) and $|\mathcal{T}|$ columns labeled with the time t.

For efficient storage and processing, we discern between static and time-varying attributes. For static-attributes a labeled array $\mathbf{S}$ maintains a row for each node $v \in V$, labeled by the node id, and a number of columns equal to the number of static attributes, labeled by the corresponding attribute name. Each row v in the array associates node v with its corresponding static attributes values. In contrast, we maintain a labeled array $\mathbf{A}_i$, for each of the time-varying attributes A_i. Rows are similarly to $\mathbf{S}$ labeled with node ids, while the columns maintain temporal information. Each cell $\mathbf{A}_i[v, t]$ maintains the value of attribute A_i of v at time t. Table 2 depicts $\mathbf{V}$, $\mathbf{S}$ and $\mathbf{A}$ for the graph of Fig. 1.

4.1 Temporal Operators Implementation

Based on our data representation, time projection is easily implemented by restricting the arrays to the columns corresponding to a given time interval.

We present in detail the algorithm implementing the union operator of a graph $G(V, E, A)$ for $\mathcal{T}_1, \mathcal{T}_2$ as depicted in Alg. 1. The algorithm takes as input the labeled arrays $\mathbf{V}, \mathbf{E}, \mathbf{S}$ and $\mathbf{A}_i$ for all time-varying attributes $A_i \in A$, and outputs the labeled arrays $\mathbf{V}_\cup, \mathbf{E}_\cup, \mathbf{S}_\cup$ and $\mathbf{A}_{i\cup}$ that maintain the information of the derived union graph $G_\cup$ for $(\mathcal{T}_1, \mathcal{T}_2)$. As our result is defined in $(\mathcal{T}_1, \mathcal{T}_2)$, we initialize all $\mathbf{V}_\cup, \mathbf{E}_\cup$ and $\mathbf{A}_{i\cup}$s as empty arrays with one column for each $t \in \mathcal{T}_1 \cup \mathcal{T}_2$ (line 1). Similarly, we restrict the input tables, except $\mathbf{S}$ that stores no temporal information, only to the columns corresponding to time $t \in \mathcal{T}_1 \cup \mathcal{T}_2$ (line 2). Union is first applied on nodes and their attributes, and then on the edges. For each node $v \in V$, the corresponding row v of $\mathbf{V}$ is accessed, and if there is a value in any $\mathbf{V}[v, t]$ equal to 1, the row is inserted in $\mathbf{V}_\cup$. The corresponding entries for the attributes of v are also updated. That is, the row corresponding to v in $\mathbf{S}$ and for each time-varying attribute A_i the corresponding rows from arrays $\mathbf{A}_i$ are retrieved and copied in $\mathbf{S}_\cup$ and $\mathbf{A}_{i\cup}$ respectively (lines 3-9). Similarly, for each edge $e \in E$ the corresponding row e of $\mathbf{E}$ is accessed, and if there is a value in any $\mathbf{E}[e, t]$ equal to 1, the row is inserted in $\mathbf{E}_\cup$ (lines 10-14).

The next operator is intersection. For selecting the nodes that are to be included in the intersection graph, that is, for a row v of $\mathbf{V}$ to be inserted into $\mathbf{V}_\cap$, we require that all elements $\mathbf{V}[v, t]$ with $t \in \mathcal{T}_1$ and $\mathbf{V}[v, t']$ with $t' \in \mathcal{T}_2$ are equal to 1. The insertion of edges is also changed accordingly. Thus, the intersection algorithm is similar to the union algorithm and only differentiates in the selection of nodes and edges to be included in the result.

Finally, for the difference operator, let $\mathcal{T}_1 - \mathcal{T}_2$, for selecting the nodes that are to be included in the difference graph, a row v of $\mathbf{V}$ is inserted into $\mathbf{V}_-$ if any $\mathbf{V}[v, t]$ with $t \in \mathcal{T}_1$ is equal to 1, and all $\mathbf{V}[v, t']$ with $t' \in \mathcal{T}_2$ are equal to 0.

4.2 Aggregation Implementation

Aggregation differentiates between distinct and non-distinct. Furthermore, to improve efficiency, static and time-varying attributes can be treated separately.

Algorithm 2 applies distinct aggregation when at least one of the attributes of aggregation is time-varying. It takes as input $\mathbf{V}, \mathbf{E}, \mathbf{S}$ and $\mathbf{A}_i$ for all time-varying attributes $A_i \in A$, interval $\mathcal{T}$ and a list L of the aggregation attributes, and outputs the labeled vectors $\mathbf{V}', \mathbf{E}'$ representing the aggregated graph G'. For each attribute A_i in L, the array $\mathbf{A}_i$ is unpivoted, i.e., its columns corresponding to the time points in $\mathcal{T}$ are appended as rows (lines 1-3), and then merged into a new array $\mathbf{A}'$ (line 4). Array $\mathbf{A}'$ is then deduplicated based on key (u, a'), where a' is a tuple formed by the aggregate attributes, so that each node u appears with a given attribute tuple a' only once (line 5). For static attributes, we restrict $\mathbf{S}$ to the columns corresponding to attributes in L (line 6) and merge the restriction result into $\mathbf{A}'$ (line 7). Groups are then formed on $\mathbf{A}'$ based on each distinct attribute tuple a' and its appearances are counted (line 8). Each such group corresponds to a node that is inserted in the aggregate graph, labeled by the attribute tuple a' and with value equal to the groups count which constitutes the node's weight (lines 9-12). To add the edges in the aggregation graph, we traverse the edge array $\mathbf{E}$ and lookup the edge ends (nodes) in $\mathbf{A}'$ to retrieve the pair of attribute tuples corresponding to the given nodes at each time point. Each such tuple pair is inserted into a new temporary array $\mathbf{A}''$ (lines 13-17). The rest of the procedure is similar to our treatment of nodes. $\mathbf{A}''$ is deduplicated based on key $(u, v), (a', a'')$, groups are formed based on pairs (a', a'') and their appearances counted (lines 18-19). Finally, for each group a new row with label (a', a'') is inserted in array $\mathbf{E}'$, representing a new edge, and with weight equal to its count that is assigned as its value (lines 20-23).

For non-distinct aggregation, Algorithm 2 is modified by omitting deduplication for both nodes and edges, thus counting all appearances of each attribute tuple, or pair of attribute tuples respectively.

Optimization is possible if aggregation concerns only static attributes. For distinct aggregation, we do not require unpivoting nor deduplication as each node has a unique value for each static attribute not depending on time. Further, for each edge the lookups required for its ends do not depend on time and again no deduplication is required. For non-distinct aggregation, as we need to count all appearances of a node, we initialize corresponding weights for each (u, a') (or $(u, v), (a', a'')$) by counting the columns in $\mathbf{V}$ (or $\mathbf{E}$) equal to 1. Then, instead of counting the appearances of each group a' (or (a', a'')), we sum their weights.

Algorithm 1: UNION

```
Input: A temporal attributed graph $G$ represented by
    $\mathbf{V}, \mathbf{E}, \mathbf{S}$ and $\{\mathbf{A}_i\}$, intervals $\mathcal{T}_1, \mathcal{T}_2$
Output: The union graph $G_\cup$ represented by
    $\mathbf{V}_\cup, \mathbf{E}_\cup, \mathbf{S}_\cup, \{\mathbf{A}_{i\cup}\}$
Initialize $\mathbf{V}_\cup, \mathbf{E}_\cup, \{\mathbf{A}_{i\cup}\}$ in $\mathcal{T}_1 \cup \mathcal{T}_2$, and $\mathbf{S}_\cup$
Restrict $\mathbf{V}, \mathbf{E}$, and $\{\mathbf{A}_i\}$ in $\mathcal{T}_1 \cup \mathcal{T}_2$
for each row $v \in \mathbf{V}$ do
    if any $\mathbf{V}[v, t] = 1$ then // $t \in \mathcal{T}_1 \cup \mathcal{T}_2$
        Insert $\mathbf{V}[v]$ in $\mathbf{V}_\cup$
        Insert $\mathbf{S}[v]$ in $\mathbf{S}_\cup$
        Insert each $\mathbf{A}_i[v]$ in the corresponding $\mathbf{A}_{i\cup}$
    end
end
for each row $e \in \mathbf{E}$ do
    if any $\mathbf{E}[e, t] = 1$ then // $t \in \mathcal{T}_1 \cup \mathcal{T}_2$
        Insert $\mathbf{E}[e]$ in $\mathbf{E}_\cup$
    end
end
return $\mathbf{V}_\cup, \mathbf{E}_\cup, \mathbf{S}_\cup, \{\mathbf{A}_{i\cup}\}$
```

Algorithm 2: DISTINCT AGGREGATION

```
Input: A temporal attributed graph $G$ represented by
    $\mathbf{V}, \mathbf{E}, \mathbf{S}$ and $\{\mathbf{A}_i\}$, intervals $\mathcal{T}$ and a list $L$ of
    aggregation attributes
Output: The aggregate graph $G'$ represented by arrays
    $\mathbf{V}'$ and $\mathbf{E}'$
for $A_i \in L$ do
    $\mathbf{A'}_i \leftarrow unpivot(\mathbf{A}_i)$
end
$\mathbf{A}' \leftarrow$ Merge all $\mathbf{A'}_i$s
$\mathbf{A}' \leftarrow \mathbf{A}'.deduplicate(v, a')$
$\mathbf{S}'$ restrict to columns for $s \in L$
$\mathbf{A}' \leftarrow$ Merge $\mathbf{A}'$ and $S$
$GroupsA' \leftarrow \mathbf{A}'.groupby(a').count()$
for each group in $GroupsA'$ do
    Insert row $a'$ in $\mathbf{V}'$
    $\mathbf{V}'[a'] \leftarrow a'.count$
end
for $(e(u, v), t) \in \mathbf{E}$ do
    $a' \leftarrow$ Look up $(u, t)$ in $\mathbf{A}'$
    $a'' \leftarrow$ Look up $(v, t)$ in $\mathbf{A}'$
    Insert $(a', a'')$ to $\mathbf{A}''$
end
$\mathbf{A}''.deduplicate((u, v), (a', a''))$
$GroupsA'' \leftarrow \mathbf{A}''.groupby((a', a'')).count()$
for each group in $GroupsA''$ do
    Insert row $(a', a'')$ to $\mathbf{E}'$
    $\mathbf{E}'[(a', a'')] \leftarrow (a', a'').count$
end
return $\mathbf{V}', \mathbf{E}'$
```

4.3 Optimizations

To implement our aggregation framework, we need to materialize all possible combinations of dimensions and all possible interval aggregations. Although, this is the best option in terms of response time of OLAP queries, generation of all possible aggregations of a large multidimensional network is quite unrealistic as it requires excessive storage space.

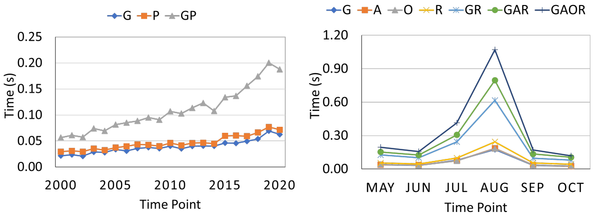
We propose using partial materialization based on the properties of the aggregate graphs and which graphs can be derived by other aggregates without accessing the original graph.

In particular, given an aggregate graph G' on a set of attributes $A' = \{A_1, A_2, \ldots, A_k\}$ any aggregate graph G'' on a subset $A'' \subseteq A'$ can be derived directly from G'. In particular, nodes and edges between them with attributes corresponding to distinct tuples in A'' are grouped together from G' to form the new aggregate nodes and edges in G''. The weights in G'' are evaluated by summing up the respective weights of the entities of G' belonging to each group. Consequently, our aggregation with COUNT as the aggregation function is identified as a *D-distributive* measure w.r.t. top-down aggregations.

With respect to time, we can compute the aggregate graph of a time interval of higher granularity based on lower level aggregate graphs if available. In particular, we claim that union and non-distinct aggregation is *T-distributive*. That is, to compute the weights of the higher level aggregate union graph, we sum up the weights for the respective nodes and edges in each of the lower level graphs. Based on the above, we propose precomputing aggregations on the unit of time that can be combined with union to attain the aggregate graphs of higher granularity. While distinct union aggregates are not T-distributive, as we need to identify distinct nodes across multiple graphs.

5 EVALUATION

Our methods are implemented in Python 3.7.9 utilizing the Modin multiprocess library [20] and our experiments are conducted in a Windows 10 machine with Intel Core i5-2430, 2.40GHz processor and 8GB RAM. We use two real-world datasets, a collaboration network, *DBLP*, and a movie ratings dataset, *MovieLens* [17]. Our code and data are publicly available[1].



(a) *DBLP*

(b) *MovieLens*

Figure 5: Aggregation time per attribute and groups of attributes per time point.

The *DBLP* dataset is extracted from DBLP[2]. Each node corresponds to an author and two authors are connected with an edge if they co-author one or more papers in a year. Our *DBLP* dataset covers a period of 21 years from 2000 to 2020 and we limit it to publications at 21 conferences related to data management research areas. The derived graph is directed and the direction of the edges indicates the order of the authors of a paper. Each node is associated with one static attribute that is the *gender* (*G*)

[1] https://github.com/etsoukanara/GraphTempo

[2] https://dblp.uni-trier.de/

Table 3: *DBLP* Dataset

#TP	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020
#Nodes	1708	2165	1761	2827	3278	4466	4730	5193	5501	5363	6236	6535	6769	7457	7035	8581	8966	9660	11037	12377	12996
#Edges	2336	2949	2458	4130	4821	7145	7296	7620	8528	8740	10163	10090	11871	12989	12072	15844	16873	18470	21197	27455	28546

Table 4: *MovieLens* Dataset

#TP	May	Jun	Jul	Aug	Sep	Oct
#Nodes	486	508	778	1309	575	498
#Edges	100202	85334	201800	610050	77216	48516

of the author and one time-varying attribute representing the number of *publications* (P) an author published each year.

The *MovieLens* dataset is built on the benchmark MovieLens [17] movie ratings dataset. We select a period of six months, from May 1st, 2000 to October 31st, 2000, where each month corresponds to a time point. Each node corresponds to a user and an edge between two users denotes that they have rated the same movie. Our dataset is directed and does not contain multiple edges in the unit of time. The order of nodes on each edge denotes precedence of rating. Each node has 3 static and 1 time-varying attribute. Specifically, we have: *gender* (G), *age* (A) that takes 6 discrete values according to the age group of a user, and *occupation* (O) that takes 21 discrete values. The time-varying attribute is the *average rating* (R) of the user per month.

Table 3 and Table 4 show nodes and edges per time point for the *DBLP* and *MovieLens* dataset respectively.

5.1 Performance Evaluation

In the first set of experiments, we evaluate the performance of our algorithms by measuring execution times.

Type of Attribute. In Fig. 5, we measure aggregation time per attribute and all attributes combinations on time points. While we expect static attributes to be faster compared to time-varying ones, in Fig. 5a, for *DBLP* time is similar for both attributes, since static and time-varying aggregation have similar behaviour when dealing with time points. Their difference is due to the larger domain of the time-varying attribute, as gender has 2 distinct values, while publications vary from 7 to 18. The aggregation (gender, publications) needs 0.18s on 2020 almost 3 times the time for gender (0.06s) and publications (0.07s). When combining the attributes, the domain gets larger, e.g. there are 28 distinct values for (gender, publications) forming the number of aggregate nodes. In Fig. 5b for *MovieLens*, for presentation clarity, we report part of all possible attribute combinations as the ones omitted behave in a similar way. Specifically, we depict the aggregation time for each attribute, and then, from all available groups with specific number of attributes, we selectively report a representative example combining static and time-varying attributes. We notice a similar to Fig. 5a pattern as gender aggregation hits the best time overall, while aggregation on all 4 attributes has the longest time. The results confirm that time is analogous to the distinct values in the aggregation attribute or combination of attributes domain. The peak observed during August is due to its high number of nodes and edges compared to other months (Table 4).

Temporal Operators. Figure 6 to 9 focus on temporal aggregations for different operators comparing static and time-varying attributes while extending time intervals. We report the total time of operation and aggregation and the time split between the temporal operator and the aggregation per attribute for *DBLP*, and total time for *MovieLens*.

Figure 6 illustrates results for union (U) and compares non-distinct (A) and distinct (D) aggregation. The type of attribute greatly influences behavior. Non-distinct aggregation for the longest interval needs 0.53s and 0.52s for gender, while 5.9s and 2.1s for the time-varying attribute for *DBLP* (Fig. 6a) and *MovieLens* (Fig. 6d) respectively. The big difference is because static attributes remain stable over time, while for time-varying ones, we need to capture all different values of the attributes in the interval. Also, distinct aggregation on static attributes achieves slightly better performance than non-distinct, up to 0.32s compared to 0.3 (Fig. 6b), as in the latter, we additionally compute the occurrences of an individual node or edge in the interval. On the contrary, non-distinct aggregation for time-varying attributes needs 5.7s while distinct is completed in 7.18s for the longest time interval (Fig. 6c). The cost for the temporal operator is similar in all cases, while aggregation increases the total time.

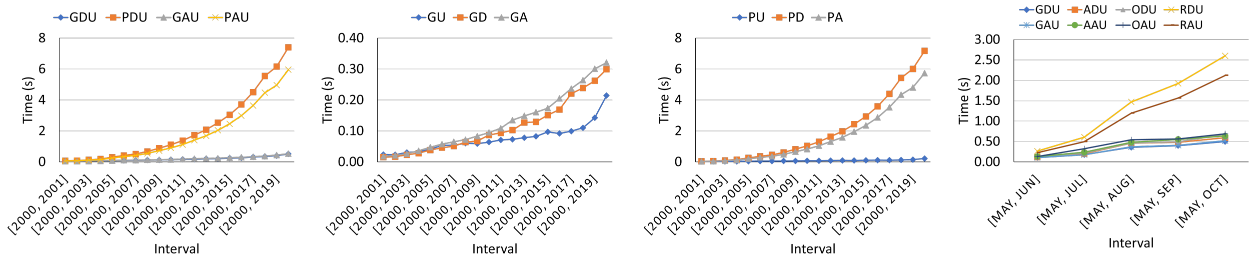
Figure 7 depicts intersection (I), where the results refer up to [2000, 2017] for *DBLP* and [May, July] for *MovieLens*, indicating the longest interval there exists at least one common edge. The type of attribute is again the controlling performance factor (Fig. 7a and Fig. 7d). In Fig. 7b, for time split, intersection is more costly than aggregation for the static attribute, in contrast to the results derived for union. This is due to the decreasing number of nodes and edges as the intervals expand. For time-varying attributes, the operation time is similar to static, but aggregation takes much longer, dominating the total cost (Fig. 7c) due to the increase in the distinct attribute values.

For difference (F), we extend $\mathcal{T}_{old}$ with union semantics, while $\mathcal{T}_{new}$ is our reference point. In Fig. 8, we consider $\mathcal{T}_{old}$ - $\mathcal{T}_{new}$. As $\mathcal{T}_{old}$ expands, total time increases as the output of the difference operation also grows. Figure 8b shows that for static attributes, similar to intersection, difference requires more than double of time required for both types of aggregation, whereas for the time-varying attribute aggregation is more expensive (Fig. 8c). For MovieLens (Fig. 8d), we notice that there is no large distinction between performance for static and time-varying attributes.

Figure 9 depicts $\mathcal{T}_{new}$ - $\mathcal{T}_{old}$, that needs less time compared to $\mathcal{T}_{old}$ - $\mathcal{T}_{new}$, as the operation output decreases. Consequently, any aggregation is faster than the operation for both static (Fig. 9b) and time-varying (Fig. 9c) attribute. Total time does not depend on the type of attribute or the type of aggregation because the operation time is about the same for both, while aggregation is actually time point aggregation.

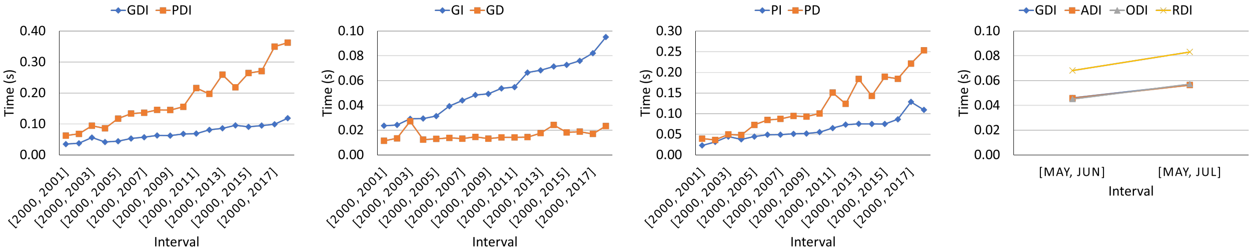
Partial Materialization. We evaluate our proposed optimizations by measuring speedup defined as the execution time of the proposed aggregation algorithms to the time of optimized aggregation that exploits precomputed results. In Fig. 10, we report speedup when exploiting precomputed aggregations on time points to derive their non-distinct union compared to computing it from scratch. For *DBLP*, aggregation over the years offers an 8x to 20x speedup for static attributes (Fig. 10a), and from 8x up to 78x for time-varying ones (Fig. 10b).

Figure 11 depicts speedup when exploiting precomputed attributes aggregations to derive aggregations on subsets of those



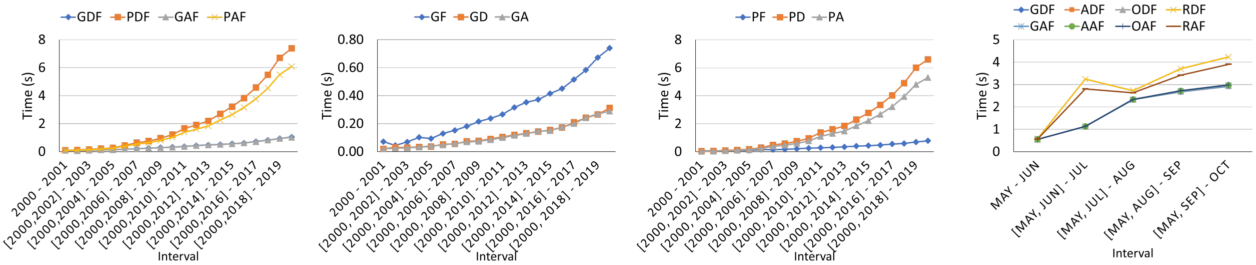
(a) Total time per attribute (b) Time split for Gender (c) Time split for #Publications (d) Total time per attribute

Figure 6: Union and aggregation (DIST, ALL) time per attribute for (a), (b), (c) *DBLP* and (d) *MovieLens*.



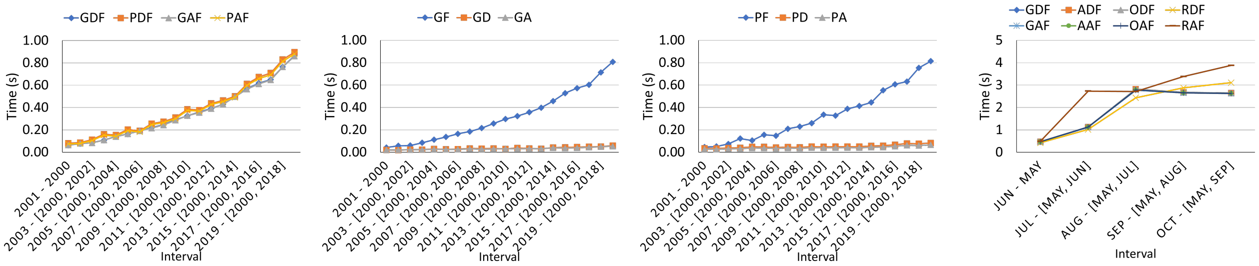
(a) Total time per attribute (b) Time split for Gender (c) Time split for #Publications (d) Total time per attribute

Figure 7: Intersection and aggregation (DIST) time per attribute for (a), (b), (c) *DBLP*, and (d) *MovieLens*.



(a) Total time per attribute (b) Time split for Gender (c) Time split for #Publications (d) Total time per attribute

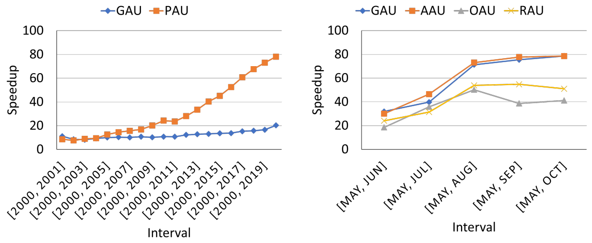
Figure 8: Difference ($\mathcal{T}_{old}(\cup)$ - $\mathcal{T}_{new}$) and aggregation (DIST, ALL) time per attribute for (a), (b), (c) *DBLP*, and (d) *MovieLens*.



(a) Total time per attribute (b) Time split for Gender (c) Time split for #Publications (d) Total time per attribute

Figure 9: Difference ($\mathcal{T}_{new}$ - $\mathcal{T}_{old}(\cup)$) and aggregation (DIST, ALL) time per attribute for (a), (b), (c) *DBLP*, and (d) *MovieLens*.

attributes compared to computing them from scratch. Figure 11a shows gender and publications aggregation when computed from the aggregation of all DBLP attributes with a speedup of 6x up to 21x. For *MovieLens*, first we report each attribute when computed from all pairs of attributes (Fig. 11b). In particular, gender is computed in $G1$ from (gender, age), in $G2$ from (gender, rating) and in $G3$ from (gender, occupation), and similarly for rating, with $R1$ from (rating, gender), $R2$ from (rating, age) and $R3$ from (rating, occupation), where we achieve a significant speedup of up to 48x. For *MovieLens*, we also report results for computing all pairs (Fig. 11c) and triplets (Fig. 11d) of attributes from the aggregate of all 4 attributes, where we observe up to 8x and up to 6x speedup, respectively. Improvement is greater for single attributes followed by pairs and then, triplets.



(a) *DBLP*

(b) *MovieLens*

Figure 10: Speedup of efficient Union aggregation (ALL).

5.2 Qualitative Evaluation

We study the evolution on gender aggregation for authors with high activity (#Publications>4) for *DBLP* where we depict the distribution of each entity w.r.t. stability, growth and shrinkage. Figure 12a depicts the aggregate evolution graph for 2010 w.r.t. the past decade, 2000's, while Fig. 12b refers to 2020 w.r.t. to 2010's. In Fig. 12a, around 61% of both male and female authors remain stable, however male authors are 86 compared to only 11 female authors. While nodes exhibit stability and there is little growth, edges have a high rate of shrinkage and no stability, so collaborations between active authors of the past decade do not continue on 2010. Figure 12b depicts similar behaviour, where the ratio of stable male authors is 9.2% higher and of women authors 17.6% higher in 2020 compared to 2010. Shrinkage in collaborations shows a similar increase.

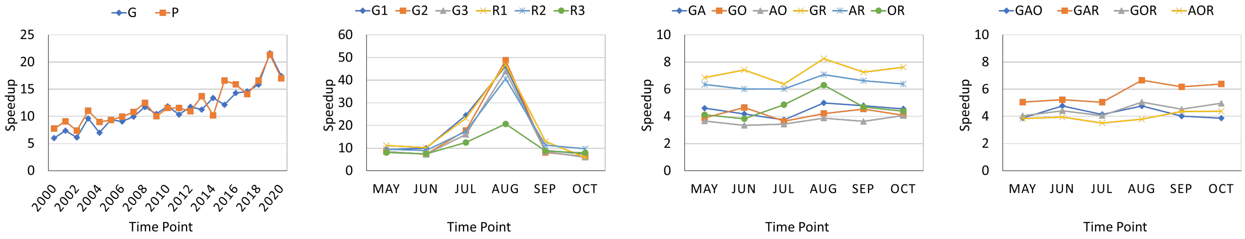
Finally, we determine maximal interval pairs for stability and minimal for growth and shrinkage for female-female relationships. For *MovieLens*, the maximum weight w_{th} is equal to 86 for stability. Thus, we gradually decrease it setting $k_3 = w_{th}$, $k_2 = w_{th}/2$ and $k_1 = w_{th}/86$. The greatest stability is achieved on August to September where there are at least 86 common edges between women (Fig. 13a). For growth, with $w_{th} = 33968$ and $k_3 = w_{th}$, $k_2 = w_{th}/2$, and $k_1 = w_{th}/12$, we report the greatest growth on August with 33968 new edges (Fig. 13b). For shrinkage, $w_{th} = 6548$ is the minimum weight and we increase it so that, $k_1 = w_{th}$, $k_2 = w_{th} * 2$, and $k_3 = w_{th} * 5$. In August, 32740 edges of [May, July] are eliminated (Fig. 13c), though August has the highest number of edges, indicating the high turnover of edges at each time point. For *DBLP* with $w_{th} = 62$, and $k_3 = w_{th}$, $k_2 = w_{th}/2$ and $k_1 = w_{th}/62$, the greatest stability occurs on 2019 with at least 62 stable collaborations (Fig. 14a). In Fig. 14b, with $w_{th} = 721$ and $k_3 = w_{th}$, $k_2 = w_{th}/3$ and $k_1 = w_{th}/10$, there is significant growth of at least 721 new collaborations on 2019. Lastly, for shrinkage, with $w_{th} = 60$ and $k_1 = w_{th}$, $k_2 = w_{th} * 5$, $k_3 = w_{th} * 20$, 1200 collaborations from 2001 up to 2009 no longer exist on 2010 (Fig. 14c).

6 RELATED WORK

In [4, 5], dimensions, measures and operators for graph OLAP are defined. Aggregate graphs are defined on both informational (attribute-based) and topological (structural) dimensions and roll-up, drill-down, slice and dice are supported. Partial materialization is discussed, while efficient computation for topological OLAP is also addressed in [21]. In [24], drill-down and roll-up are supported on graphs aggregated based on node attributes and relationships between them. GraphCube [26] also defines aggregate graphs based on both attributes and structure, where the evaluated measure is the actual aggregated graph, while it extends OLAP with queries between aggregated graphs. Graphoids are defined in [15] over labeled directed multi-hypergraphs to deal with heterogeneous data that are connected with more than binary relationships. In [9], aggregates on RDF graphs are based on SPARQL aggregation semantics, and the problem of efficiently determining the most interesting attribute-based aggregations is addressed. In [7], TigerGraph's query language defines accumulators for aggregating values returned by pattern matching. State information is gradually computed for nodes (and globally) accumulating the corresponding aggregate values for a query. In [18], aggregation is based on random walks. Each node has a score based on the concentration of the attribute values of nodes in its vicinity. The goal is to determine interesting nodes based on a user-defined threshold. In [22], a comprehensive review on graph OLAP compares related works on 7 criteria: materialization, network, selection, aggregation, model, OLAP operations and analytics. As graph aggregation and OLAP provide a concise representation of a graph, it can be also deployed for graph summarization [25]. Finally, the term graph aggregation is used in [11] to refer to the mapping of different non-attributed graphs into one, to deal with problems such as consolidating conflicting views, counting votes or computing consensus.

All works above do not focus on temporal of data. There is a lot of previous work in temporal relational databases. Temporal relational algebra [8] employs multidimensional tuple timestamping and supports both valid and transactional time. Relational operators are extended to preserve snapshot semantics, and temporal projection and temporal group by are defined. In [10], new temporal operators on sets of intervals are implemented to overcome the complexity of temporal calculations. A survey [14] on time management in data warehouses covers several issues such as warehouse design, querying and schema changes.

Little research has focused on incorporating temporal operations in the graph OLAP model. In [6], GQL is extended to T-GQL handling temporal paths. Continuous paths address evolution and consecutive paths travel scheduling problems. In TGraph [19], temporal algebraic operators such as temporal selection for nodes and edges and traversal with temporal predicates are defined for a temporal property graph. TGraph is further extended to allow aggregation on attributes and time [1], similarly to our work. Time aggregation focuses on viewing a graph in the appropriate resolution and studies stability, whereas, we focus on different events such as growth and shrinkage besides stability, and define an evolution graph that models all three through set based temporal operators. In [23], graph summarization is based on grouping nodes of the same type and aggregating on attributes and time. The system provides different visualizations of a graph, i.e., the temporal graph view, the grouped graph view where grouped elements are enriched with both attribute and temporal aggregate information, and the difference graph view that illustrates new, stable and deleted elements between two graph snapshots. In contrast to our work, the system is driven by user queries and provides no exploration strategy to determine intervals with events of interest. In [13], a conceptual model with explicit labeling of graph elements is designed to support analytical operations over evolving graphs, and particularly time-varying attributes. This model incurs redundancy by adding elements that could be discovered through graph traversal, sacrificing performance for richer analysis support. In the EvOLAP Graph [16], versioning is used, both on attributes and graph structure to enable analytics on changing graphs but no temporal operators are supported.



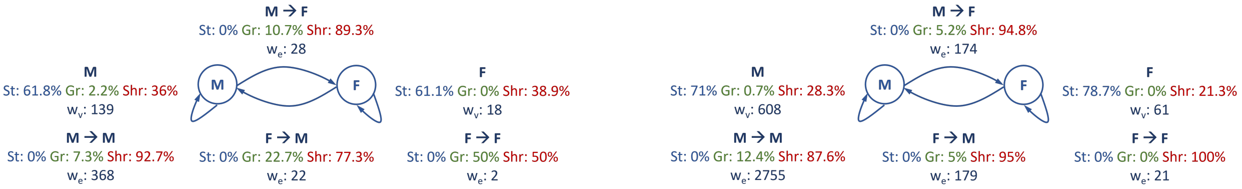
(a) Single attributes

(b) Single attributes

(c) Pairs of attributes

(d) Triples of attributes

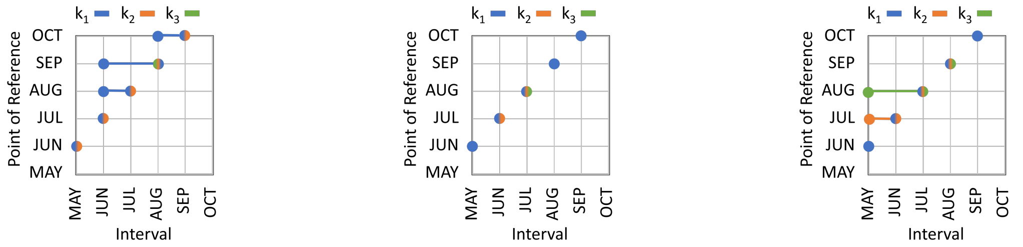
Figure 11: Speedup of efficient attribute aggregation per time point for (a) *DBLP* and (b), (c), (d) *MovieLens*.


M → F
St: 0% Gr: 10.7% Shr: 89.3%
w_e: 28
M
St: 61.8% Gr: 2.2% Shr: 36%
w_v: 139
M
F
F
St: 61.1% Gr: 0% Shr: 38.9%
w_v: 18
M → M
St: 0% Gr: 7.3% Shr: 92.7%
w_e: 368
F → M
St: 0% Gr: 22.7% Shr: 77.3%
w_e: 22
F → F
St: 0% Gr: 50% Shr: 50%
w_e: 2
M → F
St: 0% Gr: 5.2% Shr: 94.8%
w_e: 174
M
St: 71% Gr: 0.7% Shr: 28.3%
w_v: 608
M
F
F
St: 78.7% Gr: 0% Shr: 21.3%
w_v: 61
M → M
St: 0% Gr: 12.4% Shr: 87.6%
w_e: 2755
F → M
St: 0% Gr: 5% Shr: 95%
w_e: 179
F → F
St: 0% Gr: 0% Shr: 100%
w_e: 21

(a) 2010 w.r.t. 2000's

(b) 2020 w.r.t. 2010's

Figure 12: Evolution on Gender for top active authors of *DBLP*.

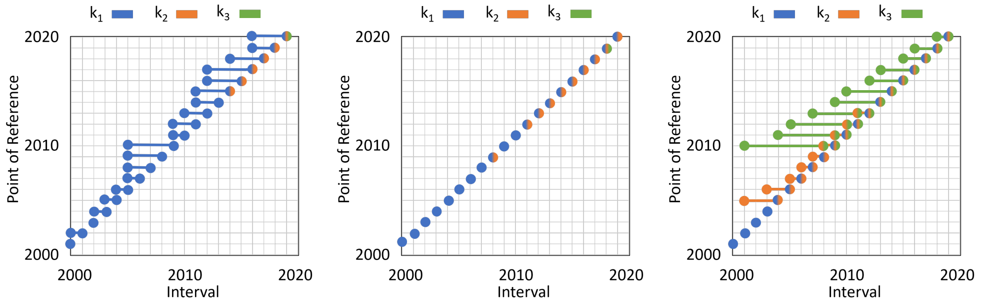


(a) Stability ($\mathcal{T}_{old}(\cap) \cap \mathcal{T}_{new}$)

(b) Growth ($\mathcal{T}_{new}$ - $\mathcal{T}_{old}(\cup)$)

(c) Shrinkage ($\mathcal{T}_{old}(\cup)$ - $\mathcal{T}_{new}$)

Figure 13: Exploration cases with (a) intersection, and (b), (c) union semantics on k for female co-rating of *MovieLens*.



(a) Stability ($\mathcal{T}_{old}(\cap) \cap \mathcal{T}_{new}$)

(b) Growth ($\mathcal{T}_{new}$ - $\mathcal{T}_{old}(\cup)$)

(c) Shrinkage ($\mathcal{T}_{old}(\cup)$ - $\mathcal{T}_{new}$)

Figure 14: Exploration cases with (a) intersection, and (b), (c) union semantics on k for female collaborations of *DBLP*.

7 CONCLUSIONS

In this paper, we introduce GraphTempo, a model designed with a twofold purpose (a) the temporal aggregation of attributed graphs and (b) the exploration of their evolution. We define a set of temporal operators for considering graphs on intervals of varying granularity with both tight and relaxed semantics. We also define a novel structure that captures graph evolution, and show how we can explore it to determine the minimal intervals with significant growth and shrinkage or the maximal intervals with significant stability. We experimentally evaluate the cost of the proposed operators and showcase the evolution of two real datasets. We plan to develop GraphTempo into an interactive exploration framework that will assist users navigate large graphs and detect intervals and attribute groups of interest in their evolution.

REFERENCES

[1] Amir Aghasadeghi, Vera Zaychik Moffitt, Sebastian Schelter, and Julia Stoyanovich. 2020. Zooming Out on an Evolving Graph. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020.* OpenProceedings.org, 25–36.

[2] Jaewook Byun, Sungpil Woo, and Daeyoung Kim. 2020. ChronoGraph: Enabling Temporal Graph Traversals for Efficient Information Diffusion Analysis over Time. *IEEE Trans. Knowl. Data Eng.* 32 (2020), 424–437.

[3] Alexander Campos, Jorge Mozzino, and Alejandro A. Vaisman. 2016. Towards Temporal Graph Databases. *CoRR* abs/1604.08568 (2016).

[4] Chen Chen, Xifeng Yan, Feida Zhu, Jiawei Han, and Philip S. Yu. 2008. Graph OLAP: Towards Online Analytical Processing on Graphs. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy.* 103–112.

[5] Chen Chen, Xifeng Yan, Feida Zhu, Jiawei Han, and Philip S. Yu. 2009. Graph OLAP: a multi-dimensional framework for graph data analysis. *Knowl. Inf. Syst.* 21, 1 (2009), 41–63.

[6] Ariel Debrouvier, Eliseo Parodi, Matías Perazzo, Valeria Soliani, and Alejandro A. Vaisman. 2021. A model and query language for temporal graph databases. *VLDB J.* 30, 5 (2021), 825–858.

[7] Alin Deutsch, Yu Xu, Mingxi Wu, and Victor E. Lee. 2020. Aggregation Support for Modern Graph Analytics in TigerGraph. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20).* Association for Computing Machinery, New York, NY, USA, 377–392. https://doi.org/10.1145/3318464.3386144

[8] Debabrata Dey, Terence M. Barron, and Veda C. Storey. 1996. A Complete Temporal Relational Algebra. *VLDB J.* 5, 3 (1996), 167–180.

[9] Yanlei Diao, Pawel Guzewicz, Ioana Manolescu, and Mirjana Mazuran. 2021. Efficient Exploration of Interesting Aggregates in RDF Graphs. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021.* ACM, 392–404.

[10] Andreas Dohr, Christiane Engels, and Andreas Behrend. 2018. Algebraic Operators for Processing Sets of Temporal Intervals in Relational Databases. In *25th International Symposium on Temporal Representation and Reasoning, TIME 2018, Warsaw, Poland, October 15-17, 2018*, Vol. 120. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 11:1–11:16.

[11] Ulle Endriss and Umberto Grandi. 2017. Graph aggregation. *Artificial Intelligence* 245 (2017), 86–114.

[12] Valerio Gemmetto, Alain Barrat, and Ciro Cattuto. 2014. Mitigation of infectious disease at school: targeted class closure vs school closure. *BMC Infectious Diseases* 14 (2014), 695.

[13] Amine Ghrab, Sabri Skhiri, Salim Jouili, and Esteban Zimányi. 2013. An Analytics-Aware Conceptual Model for Evolving Graphs. In *Data Warehousing and Knowledge Discovery - 15th International Conference, DaWaK 2013, Prague, Czech Republic, August 26-29, 2013. Proceedings*, Vol. 8057. 1–12.

[14] Matteo Golfarelli and Stefano Rizzi. 2009. A Survey on Temporal Data Warehousing. *Int. J. Data Warehous. Min.* 5, 1 (2009), 1–17.

[15] Leticia I. Gómez, Bart Kuijpers, and Alejandro A. Vaisman. 2020. Online analytical processsing on graph data. *Intell. Data Anal.* 24, 3 (2020), 515–541.

[16] Ewa Guminska and Teresa Zawadzka. 2018. EvOLAP Graph - Evolution and OLAP-Aware Graph Data Model. In *Beyond Databases, Architectures and Structures. Facing the Challenges of Data Proliferation and Growing Variety - 14th International Conference, BDAS 2018, Held at the 24th IFIP World Computer Congress, WCC 2018, Poznan, Poland, September 18-20, 2018, Proceedings (Communications in Computer and Information Science)*, Vol. 928. Springer, 75–89.

[17] F. Maxwell Harper and Joseph A. Konstan. 2016. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4 (2016), 19:1–19:19.

[18] Nan Li, Ziyu Guan, Lijie Ren, Jian Wu, Jiawei Han, and Xifeng Yan. 2013. gIceberg: Towards iceberg analysis in large graphs. In *2013 IEEE 29th International Conference on Data Engineering (ICDE).* 1021–1032. https://doi.org/10.1109/ICDE.2013.6544894

[19] Vera Zaychik Moffitt and Julia Stoyanovich. 2017. Temporal graph algebra. In *Proceedings of The 16th International Symposium on Database Programming Languages, DBPL 2017, Munich, Germany, September 1, 2017.* ACM, 10:1–10:12.

[20] Devin Petersohn, William W. Ma, Doris Jung Lin Lee, Stephen Macke, Doris Xin, Xiangxi Mo, Joseph Gonzalez, Joseph M. Hellerstein, Anthony D. Joseph, and Aditya G. Parameswaran. 2020. Towards Scalable Dataframe Systems. *Proc. VLDB Endow.* 13, 11 (2020), 2033–2046.

[21] Qiang Qu, Feida Zhu, Xifeng Yan, Jiawei Han, Philip S. Yu, and Hongyan Li. 2011. Efficient Topological OLAP on Information Networks. In *Database Systems for Advanced Applications - 16th International Conference, DASFAA 2011, Hong Kong, China, April 22-25, 2011, Proceedings, Part I (Lecture Notes in Computer Science)*, Vol. 6587. 389–403.

[22] Paulo Orlando Queiroz-Sousa and Ana Carolina Salgado. 2020. A Review on OLAP Technologies Applied to Information Networks. *ACM Trans. Knowl. Discov. Data* 14, 1 (2020), 8:1–8:25.

[23] Christopher Rost, Kevin Gómez, Philip Fritzsche, Andreas Thor, and Erhard Rahm. 2021. Exploration and Analysis of Temporal Property Graphs. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021.* OpenProceedings.org, 682–685.

[24] Yuanyuan Tian, Richard A. Hankins, and Jignesh M. Patel. 2008. Efficient aggregation for graph summarization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008.* 567–580.

[25] Šejla Čebirić, François Goasdoué, Haridimos Kondylakis, Dimitris Kotzinos, Ioana Manolescu, Georgia Troullinou, and Mussab Zneika. 2019. Summarizing Semantic Graphs: A Survey. *The VLDB Journal* 28, 3 (jun 2019), 295–327. https://doi.org/10.1007/s00778-018-0528-3

[26] Peixiang Zhao, Xiaolei Li, Dong Xin, and Jiawei Han. 2011. Graph cube: on warehousing and OLAP multidimensional networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011.* ACM, 853–864.