THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse Capitole (UT Capitole)*

Présentée et soutenue le *6 Décembre 2023* **par :**

Landy ANDRIAMAMPIANINA

Temporal Graphs: From Modelling to Analysis

JURY

Laurent WENDLING	Professeur, Université Paris Cité	Rapporteur
Esteban ZIMÁNYI	Professeur, Université Libre de Bruxelles	Rapporteur
Arnaud CASTELLTORT	Maître de Conférences, Polytech Montpellier	Examinateur
Chantal SOULÉ-DUPUY	Professeure, Université Toulouse Capitole	Présidente du jury
Franck RAVAT	Professeur, Université Toulouse Capitole	Directeur de thèse
Nathalie VALLÈS-PARLANGEAU	Maîtresse de Conférences, Université de Pau et des Pays de l'Adour	Co-directrice de thèse

École doctorale et spécialité :

EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse : Informatique et Télécommunications

Unité de Recherche :

IRIT: Institut de Recherche en Informatique de Toulouse (UMR 5505)

Directeur(s) de Thèse :

Franck RAVAT et *Nathalie VALLÈS-PARLANGEAU*

Rapporteurs :

Laurent WENDLING et *Esteban ZIMÁNYI*

À mon époux Toky

Acknowledgement

I would like to express my deep and sincere gratitude to my supervisor, **Professor Franck RAVAT**, and to my co-supervisor, **Doctor Nathalie VALLÈS-PARLANGEAU**, for guiding me throughout this thesis. Ever since I became their Master's student, they have always challenged me to surpass myself, not least by giving me the opportunity to do this thesis with them. Throughout this thesis, their passion for research, rigour, patience, encouragement and above all their belief in my potential have been a constant source of motivation. I deeply value their humility, generosity, comprehensive support and good humour, which have comforted me in difficult times. I am also deeply grateful to them for not hesitating to sacrifice their personal time to help me when I needed it.

I would like to extend my heartfelt gratitude to my former supervisor in enterprise, **Doctor Jiefu SONG**, for providing me the opportunity to join the *Activus GROUP* company and delve into a captivating research subject. I am deeply appreciative of his exceptional guidance, which has allowed me to make continuous progress on my research, and his careful revision of my work has led my achievements to a higher quality. His constant support played a vital role in my integration into the company and the research community. I would also like to thank him for his natural positivity, attentive ear, and great patience, which guided me through moments of uncertainty and helped me regain my self-confidence.

I would like to express my deep appreciation to all committee members for making my defense an enjoyable moment in my life. My sincere thanks goes to my thesis reviewers **Professor Laurent WENDLING** and **Professor Esteban ZIMÁNYI** for evaluating and approving my work. Their attentive reviews, valuable comments, and enlightening questions have contributed greatly to improve my work.

I would like to thank my thesis examiners, **Doctor Arnaud CASTELLTORT** and **Professor Chantal SOULÉ-DUPUY**, for their valuable comments, shared ideas and helpful suggestions.

I would like to acknowledge **CEO Pascal DIOGO** of *Activus GROUP* and the **French National Association for Research and Technology (ANRT)** for funding my thesis (with reference number 2019/0969), offering me the opportunity and giving me a lot of autonomy for my scientific research.

Special thanks to the interns **Abudujilili AERSILANG**, **Zijian LI**, **Ruohan REN** and **Yanpei WANG**, my colleagues **Zhi CHENG**, **Pierre-Paul CAVALLERA**, and **Philippe RENAUDIN**, and my friends **Vincent-Nam DANG**, **Bastien VIDÉ**, **Haomiao WANG** and **Edouard ZONZA**, whose help have been invaluable in advancing my research.

My sincere thanks also go to **all my colleagues** at *Université Toulouse Capitole* and *Activus GROUP*, for sharing the adventure of this thesis. Their company and support provided me with a pleasant working atmosphere.

I am extremely grateful to **my parents**, José and Voahirana, **my brother**, Rado, and **my grandmother**, Odette, for their love, encouragement and for all sacrifices they have made. My love and gratitude for them can hardly be expressed in words.

I would also like to acknowledge **my family in France**, Masy, Benoît, Marius, Arthur, Liana, Jordan, Aro, Julia, Rialy, Edouard, Kévin, Anaïs, Valentin A., Laurène, Steeve, Océane, Brian, Marion, Valentin G., Larry, Maeva, Julien and Mihary, for your endless help every day.

Finally, I would like to thank **my beloved husband Toky**, to whom this thesis is dedicated. Your love, care, patience and never-ending support has helped me to never give up. Thank you for sharing your life with me and for bringing me happiness every day.

Résumé

Aujourd'hui, les entités du monde réel sont de plus en plus interconnectées (par exemple, les individus qui interagissent sur les plateformes sociales). Néanmoins, ces entités et leur interconnectivité évoluent continuellement dans le temps. Elles peuvent apparaître et disparaître au fil du temps. De plus, leurs caractéristiques descriptives peuvent être ajoutées, supprimées ou mises à jour au fil du temps.

Les données générées par les entités interconnectées sont généralement représentées par des *Graphes*. Cependant, les graphes statiques ne sont pas suffisants pour intégrer le concept d'évolution temporelle. Cette thèse aborde donc la question de recherche suivante : Comment permettre des analyses sur des données graphes enrichies par des évolutions temporelles ? Ce problème implique trois défis principaux : (i) Comment incorporer l'évolution temporelle dans un graphe statique ?, (ii) Comment trouver des informations dans un graphe temporel ? et, (iii) Comment découvrir des connaissances dans un tel graphe ?

Du point de vue de la modélisation, nous définissons une solution de gestion complète pour les graphes intégrant des évolutions temporelles : un modèle conceptuel jusqu'à sa mise en œuvre. Tout d'abord, notre modèle conceptuel, appelé *Graphe Temporel*, inclut des concepts proches du monde réel : des entités, des relations et des états pour capturer leur évolution temporelle. Deuxièmement, nous proposons des règles pour traduire automatiquement notre modèle conceptuel en un modèle de graphe de propriétés. Troisièmement, nous proposons une implémentation dans des solutions de stockage orientées graphe. Enfin, les résultats expérimentaux montrent que notre solution de gestion est (i) réalisable, c'est-à-dire implémentable dans une solution de stockage orientée graphe, (ii) utilisable pour les analyses orientées métier, (iii) efficace en termes de stockage et de performance d'interrogation, et (iv) passant à l'échelle lorsque le volume de données augmente.

Du point de vue de l'interrogation, nous fournissons une solution permettant aux utilisateurs de trouver des informations pour répondre à des questions métiers (*'Quoi ?', 'Qui ?', 'Où ?', 'Quand ?'*). L'avantage de notre solution d'interrogation pour les graphes avec des évolutions temporelles est d'être complète. Tout d'abord, cette solution inclut des opérateurs conceptuels, qui sont orientés vers l'utilisateur et composables. Ils permettent de trouver des informations dépendantes du temps sur la topologie, ainsi que sur différents composants d'un graphe temporel. Pour être applicables, nous proposons des règles de traduction de nos opérateurs conceptuels vers des opérateurs logiques pour l'interrogation du modèle de graphe de propriétés. Des expérimentations sont menées pour vérifier que notre solution d'interrogation permet d'appliquer efficacement des analyses orientées métier sur des jeux de données du monde réel.

Du point de vue de la découverte de connaissances, nous proposons une solution permettant aux utilisateurs d'extraire des informations cachées pour répondre à des questions complexes (*'Comment ?'*). D'une part, cette solution définit un nouveau motif, spécifiant une combinaison d'éléments d'information d'un graphe temporel à extraire. Notre motif présente l'avantage (i) de capturer pleinement l'information des multiples dimensions d'un graphe temporel et (ii) de représenter des mécanismes d'évolution couvrant plusieurs groupes d'entités connectées au lieu d'un seul. D'autre part, nous proposons un algorithme permettant d'extraire notre motif d'un graphe temporel. Étant donné que tous les algorithmes d'extraction de motifs sont confrontés au problème de la complexité de calcul élevée, nous proposons une stratégie d'extraction pour réduire cette dernière. Nous menons des expérimentations confirmant que (i) notre motif est utile, notamment pour comprendre les impacts d'événements perturbateurs dans des jeux de données du monde réel, et que (ii) notre algorithme passe à l'échelle lorsque le volume de données augmente.

Abstract

Today, real-world entities are becoming increasingly interconnected (e.g., individuals interacting on social platforms). Nevertheless, these entities and their interconnectivity evolve continually over time. They may appear and disappear over time. Moreover, their descriptive characteristics may be added, removed or updated over time.

Data generated by interconnected entities are generally represented by *Graphs*. However, static graphs are not enough to integrate the concept of temporal evolution. This thesis addresses therefore the problem of enabling analyses on graph data enriched with temporal evolution. This problem induces three main challenges: (i) How can we incorporate temporal evolution in a static graph?, (ii) How can we find information in a temporal graph? and, (iii) How can we discover knowledge in such a graph?

From a modelling point of view, we define a complete management solution for graphs with temporal evolution, from a conceptual model to its implementation. First, our conceptual model, called *Temporal Graph*, includes concepts close to the real-world: entities, relationships, and states to capture their temporal evolution. Second, we propose mapping rules to translate automatically our conceptual model to the property graph model. Third, we propose an implementation in graph-oriented data stores. Finally, the experimental results show that our management solution is (i) feasible, i.e., implementable in graph-oriented data stores, (ii) usable for business analyses, (iii) efficient in terms of storage and query performance, and (iv) scalable when the data volume increases.

From a querying point of view, we provide a solution allowing users to find information for answering business questions (*'What?', 'Who?', 'Where?', 'When?'*). The advantage of our querying solution for graphs with temporal evolution is to be complete. First, this solution includes conceptual operators, which are user-oriented and composable. They enable to find time-dependent information on the topology, as well as on different components of the temporal graph. To be implementable, we propose mapping rules of our conceptual operators into logical operators for querying the property graph model. We verify through experiments that our querying solution allows effectively applying business analyses on real-world datasets.

From a knowledge discovery point of view, we offer a solution allowing users to extract hidden information for answering complex business questions (*'How?'*). On the one hand, this solution defines a novel pattern, specifying a combination of information pieces of temporal graph to be extracted. Our pattern has the advantages of (i) fully capturing information from the multiple dimensions of a temporal graph and (ii) representing evolution mechanisms spanning several groups of connected entities instead of a single one. On the other hand, we propose an algorithm to extract our pattern from a temporal graph.

Since all pattern mining algorithms face the problem of high computational complexity, we propose a mining strategy to reduce the latter. We conduct experiments confirming that (i) our pattern is useful, notably to understand the impacts of disruptive events in real-world datasets, and that (ii) our algorithm is scalable when data volume increases.

Publication List

National Conferences

Andriamampianina, L., Ravat, F., Song, J., & Vallès-Parlangeau, N. (2020). A generic modelling to capture the temporal evolution in graphs. In 16e jounées EDA Business Intelligence & Big Data (EDA), vol. RNTI-B-16, pp.19-32

International Conferences

Andriamampianina, L., Ravat, F., Song, J., & Vallès-Parlangeau, N. (2021). Towards an efficient approach to manage graph data evolution: conceptual modelling and experimental assessments. In International Conference on Research Challenges in Information Science (pp. 471-488). Springer, Cham.

Andriamampianina, L., Ravat, F., Song, J., & Vallès-Parlangeau, N. (2022). Querying Temporal Property Graphs. In International Conference on Advanced Information Systems Engineering (pp. 355-370). Springer, Cham.

Cheng, Z., Andriamampianina, L., Ravat, F., Song, J., Vallès-Parlangeau, N., Fournier-Viger, P., & Selmaoui-Folcher, N. (2023). Mining Frequent Sequential Subgraph Evolutions in Dynamic Attributed Graphs. In Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer.

Andriamampianina, L., Ravat, F., Song, J., & Vallès-Parlangeau, N. (2023). Semantic Centrality for Temporal Graph. In New Trends in Database and Information Systems - ADBIS 2023 Short Papers, Barcelona, Spain, September 4-7, 2023, Proceedings.

International Journals

Andriamampianina, L., Ravat, F., Song, J., & Vallès-Parlangeau, N. (2022). Graph data temporal evolutions: From conceptual modelling to implementation. Data & Knowledge Engineering, 139, 102017.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1 Context

The interconnectivity of real-world entities is a fundamental aspect of our contemporary society. From the physical to the digital sphere of our society, we observe relationships, interactions between entities everywhere. In the physical world, biological ecosystems connect living organisms, transportation systems link cities, social systems (workplaces, education, marketplaces) connect individuals, and so on. The widespread adoption of digital technologies and platforms in our society has extended the interconnectivity of real-world entities to the virtual realm. Information systems in organizations connect employees, IoT devices connect physical objects of our daily lives (e.g., smart lights, smart TVs), social platforms (e.g., Meta, LinkedIn) connect individuals, online marketplaces (e.g., Amazon) connect consumers and sellers, and so on. Besides, our society is becoming more interconnected by the day. According to the International Data Corporation, while the number of interactions per digitally connected person per day worldwide was 218 in 2015, it is expected to be 20 times more in 2025, up to 4 758 (Reinsel et al., 2017).

Data generated everywhere by interconnected entities has led organizations to consider a new form of data beyond the conventional forms (e.g., tabular data, relational data, textual data). Data is no longer perceived as isolated pieces, but as interconnected pieces. In this context, the concept of *Graphs*, a collection of vertices connected by edges, has appeared as a natural way to represent real-world entities connected by relationships (or interactions) (Angles and Gutierrez, 2018). Moreover, graphs unlock fresh analytical perspectives of real-world scenarios by enabling the extraction of valuable information and the discovery of hidden information from the relationships they encapsulate (Ghrab et al., 2018). Gartner estimates that by 2025, graph technologies will be used in 80% of data and analytics innovations, which allows facilitating decision-making across organizations (Adrian and Jaffri, 2022). Currently, the ACTIVUS Group[1] company, the industrial financial partner of this Ph.D. thesis, leverages the power of graphs to represent its clients' needs within its software solution and to provide graph-oriented analysis possibilities (e.g., retrieve linking chains between objects of networks).

Classic graphs are static and thus not enough in some real-world cases, since entities and their relationships continually evolve over time. New entities may appear and disappear over time. On online marketplaces, new sellers, products and consumers may regularly enter or leave the market. Similarly, the relationships between entities may appear and disappear over time. In social networks, individuals can add or delete friends. The descriptive characteristics of entities may also change over time. In workplaces, information systems record changes in their employee career changes (e.g. new skills, promotions, new positions). Similarly, the descriptive characteristics of relationships between entities may change over time. In transportation systems, the speed limit on a road segment between two intersections might undergo changes over time. To meet these needs, it is therefore important to integrate the concept of temporal evolution in Graphs. This opens up new prospects for ACTIVUS Group and new research perspectives. This Ph.D. thesis falls within this context.

[1] https://www.activus-group.fr/

2 Problem Definition

In response to the needs expressed in Section 1, we address in this thesis the following research question: How can we enable analyses on graph data enriched with temporal evolution? To answer this question, we identify the following scientific challenges.

2.1 How can we incorporate temporal evolution in static graphs ?

To incorporate the temporal evolution into graphs, we must address several fundamental questions. First, as discussed before, in the real-world, temporal evolution may occur at the level of entities, relationships and their descriptive characteristics. Consequently, it requires determining which levels within the graph are subject to this temporal evolution. In simpler terms, how do we associate the concept of evolution to the graph's components, such as vertices and edges? Next, we need to consider the abstraction levels that the graph can embody while accounting for temporal evolution. These levels range from closely mirroring the real-world to more technically oriented representations. It is important to establish where the graph falls within this spectrum, as it influences how we interpret temporal changes in the graph. Lastly, how can we manage the data changes in the graph? Is it better to keep current and past graph data together, or to keep certain states of the graph over time?

In the existing literature, some research works have partially answered these challenges. They associate temporal evolution only on specific levels within the graph (Zaki et al., 2016). Moreover, their approaches tend to be more technically oriented and may diverge significantly from the representation of the real-world (Kosmatopoulos et al., 2016). Finally, to manage data changes over time, they often rely on the use of graph snapshots taken at different points in time. However, this method prevents for having direct access to information about the changes in graph data during analyses (Moffitt and Stoyanovich, 2017b).

2.2 How can we explore graphs with temporal evolution?

Users can explore a graph with temporal evolution to answer various questions about entities of a business context, connected by relationships and evolving over time. As starting point of this exploration, they may want to find information in the graph with temporal evolution to answer simple business-oriented questions : *'What?', 'Who?', 'Where?', 'When?'*. To go further in the exploration, they may want to extract knowledge to address more complex questions: *'How ?'* questions. In this context, knowledge refers to a combination of information pieces that serves for understanding causality or correlations of phenomena (or events) within the graph with temporal evolution and for helping decision-making (Bellinger et al., 2004; Rowley, 2007). The main challenge for knowledge extraction in a graph with temporal evolution is to be able to combine information from its multiple dimensions (evolution, topology, etc.) and other external information to obtain some added-value (Fournier-Viger et al., 2020a).

In the existing literature, solutions have been designed on the one hand to find information in graphs with temporal evolution. However, these solutions are often insufficient in terms of their ability to manipulate the diverse dimensions of graphs with temporal evolution. Moreover, they tightly dependent on specific technical environments and lack of a user-oriented framework (Zhang et al., 2019; Debrouvier et al., 2021). On the other hand, existing solutions for extracting knowledge in graphs with temporal evolution struggle to fully capture the information from their several dimensions. This limitation diminishes their capacity to provide comprehensive explanations about users' business contexts.

3 Contributions

In response to our research question presented in Section 2, the objective of this thesis is to propose solutions enabling analyses on graphs with temporal evolution. These solutions are intended to be as generic as possible in order to serve a wider range of users and applications. More precisely, we address the challenges presented in Section 2 as follows.

To enhance static graphs with temporal evolution, we propose a complete management solution of graphs with temporal evolution, from their modelling to their implementation. We introduce a conceptual (i.e., business-oriented) model, called *Temporal Graph* (TG), which offers a means to represent real-world entities, relationships, and their evolution across all levels. Our conceptual model associates evolution to each graph component to enable the direct access to information about changes. Moreover, we propose an implementation framework of our TG model to facilitate its translation into technical environments.

To explore graphs with temporal evolution, we make two propositions. On the one hand, we propose a complete querying solution for finding information in graphs with temporal evolution, from a conceptual to an implementation framework. At the conceptual level, we propose *conceptual operators* providing the ability to fully manipulate the diverse dimensions of a graph with temporal evolution to formulate business questions. The implementation framework guarantees the easy translation of these conceptual operators into several alternatives of technical environments.

On the other hand, we offer a solution for knowledge extraction in a graph with temporal evolution. On the one hand, this solution consists of a *pattern* allowing to specify the information pieces of a graph with temporal evolution to be combined to obtain knowledge. This pattern fully captures the information from the multiple dimensions of the graph with temporal evolution. On the other hand, it consists of an algorithm to extract the pattern from the graph with temporal evolution.

Through this thesis, we have developed feasibility and efficiency studies in real-world contexts. They will serve as a proof-of-concept to integrate a temporal layer at the level of data representation and analysis of the graph-oriented software of ACTIVUS Group.

4 Manuscript Outline

The manuscript is organized as follows:

- In Chapter 2, first, we make a literature review on current models of graphs with temporal evolution and their implementation. Second, we present our conceptual model of TG. Third, we present the implementation framework of our conceptual model of TG. Fourth, we evaluate the feasibility and usability of our TG model through the analysis of a case study. Finally, we make some experiments to evaluate (i) the efficiency of the implementation of our TG model compared to classic TG models, and (ii) the scalability of the implementation of our TG model across different data volumes.

- In Chapter 3, first, we make a literature review on existing querying solutions for graphs with temporal evolution. Second, we present our conceptual operators for querying graphs with temporal evolution. Third, we present the implementation framework of our conceptual operators through two abstraction levels: logical and physical. Finally, we make experiments to verify the feasibility of this querying solution using several benchmark queries over different application domains.

- In Chapter 4, first, we make a literature review on solutions in the Pattern Mining field. This field proposes analytical techniques for extracting patterns (i.e., combination of information pieces) in graphs with temporal evolution using algorithms. Second, we define a novel pattern and show its advantage compared to existing patterns. Third, we present the algorithm we propose to extract our pattern from a graph with temporal evolution similar to our TG model. Finally, we conduct experiments to evaluate the scalability of the algorithm across different data volumes and to evaluate the interest of the extracted patterns across different application domains.

We illustrate in Fig 1.1 the full scope of this thesis.

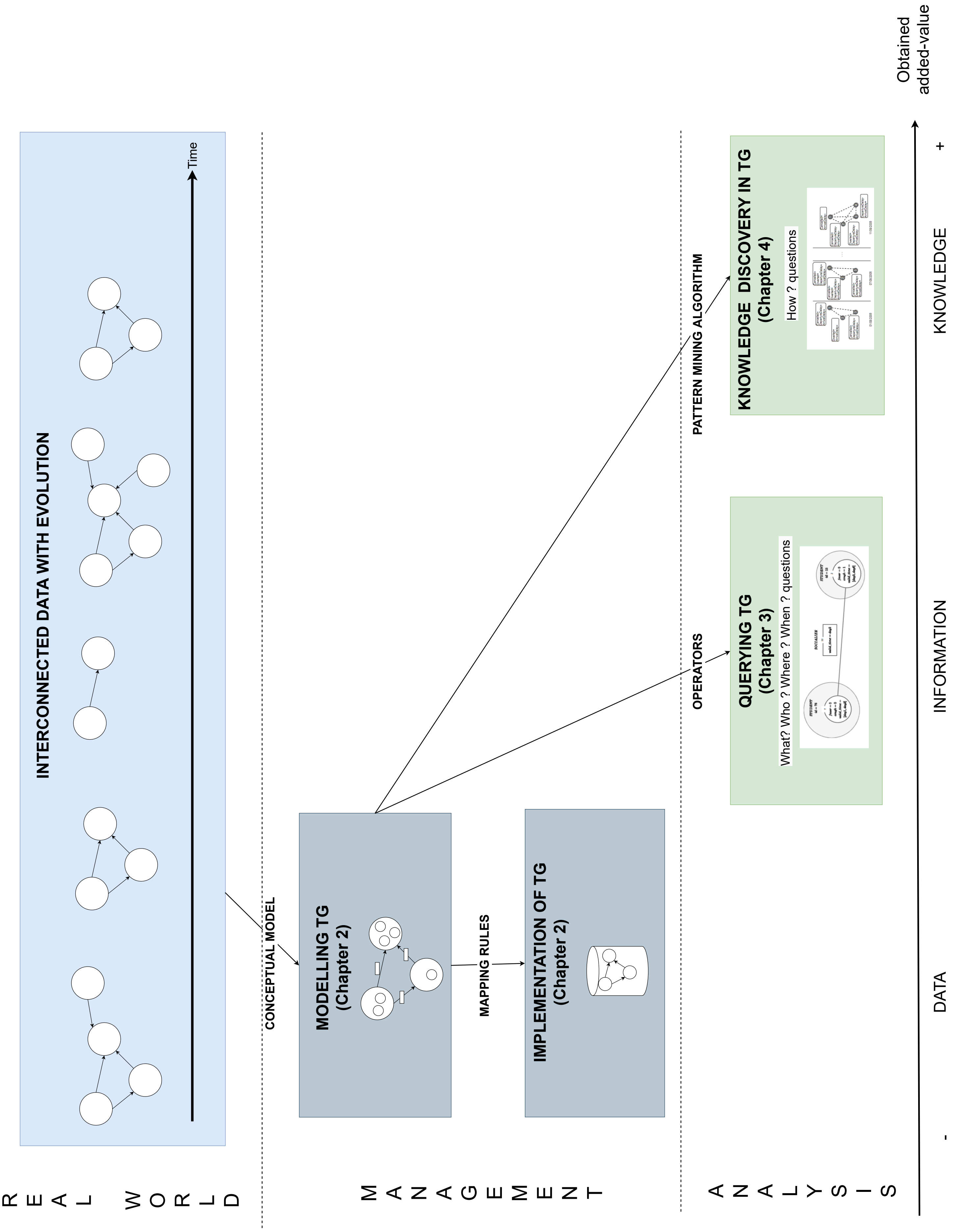


Figure 1.1: Thesis Scope

Chapter 2

Modelling Temporal Graphs

Contents

1 Introduction

Data modelling is a fundamental step for exploring interconnected data evolving over time (Angles and Gutierrez, 2008). It serves the purpose of structuring and defining the semantics of data. Traditionally, interconnected data are modelled using the concept of *Graphs*, i.e., a collection of vertices connected by edges, as they naturally represent the connections within data (Angles and Gutierrez, 2008). In the literature, several data models extend Graphs to integrate the concept of temporal evolution. These data models go by many different names such as *'Temporal Graphs', 'Evolving Graphs', 'Time-varying Graphs' or 'Dynamic Graphs',* among others (Holme and Saramäki, 2012; Debrouvier et al., 2021). For simplicity, we will use the term 'Temporal Graphs' (TG) to refer to data models that handle graphs with temporal evolution.

Current TG models come with certain limitations that hinder their use for data exploration purposes by users. First, current TG models associate temporal evolution to specific levels of the graph to meet the requirements of specific applications: the topology to capture the addition and removal of connections between data pieces, the attribute value of vertices or edges to capture the update of data values, or the combination of these different levels (Yang et al., 2014; Rossi et al., 2013; Aslay et al., 2018; Desmier et al., 2012; Latapy et al., 2018; Zhao et al., 2020; Debrouvier et al., 2021; Campos et al., 2016). As they are application-specific, they may not accommodate the diverse types of changes required by various users and applications. Second, they generally keep track of the changes in graph data by representing snapshots of a graph over different points in time (Zaki et al., 2016). This tracking strategy does not allow direct access to information about evolution at different graph levels (e.g., at the level of a vertex). Finally, current TG models tend to be more technically oriented by considering implementation issues (Kosmatopoulos et al., 2016). This limits their usability for real-world scenarios.

After modelling interconnected data evolving over time, it is necessary to implement a TG data model into technical environments to effectively explore them. This implementation begins with a data storage infrastructure adapted for temporal graph data. This ensures that data is organized and accessible for further use. Generally, the implementation of current TG models is tailored to particular applications and dependent on specific technical environments (Ren et al., 2011; Khurana and Deshpande, 2013, 2016; Gandhi and Simmhan, 2020; Ramesh et al., 2020; Xiangyu et al., 2020; Cattuto et al., 2013; Huang et al., 2016). In a nutshell, current TG models are not generic enough either in terms of evolution representation or implementation environments, to meet the needs of a wide audience of users and applications.

In this chapter, the objective is to provide a complete management solution of temporal graph data, ranging from a modelling to an implementation (Fig 2.1). The model we propose must associate temporal evolution to all levels of the graph. Moreover, it must propose a tracking strategy that allows direct access to information at all graph levels. Finally, it should provide business-oriented concepts to align closely to real-world scenarios. To do so, first, we review the literature on existing models of TG and their implementation (Section 3). Second, we propose a conceptual model of TG and a graphical notation to facilitate its exploitation by non-expert users (Section 4). Third, we propose mapping rules to translate our conceptual model into the logical property-graph model that is supported by several technical environments (Section 5). Fourth, we implement our conceptual model to evaluate the feasibility and usability through business analyses (Section 6). Finally, we evaluate the efficiency and scalability of our solution through experiments using benchmark and real-world datasets (Section 7).

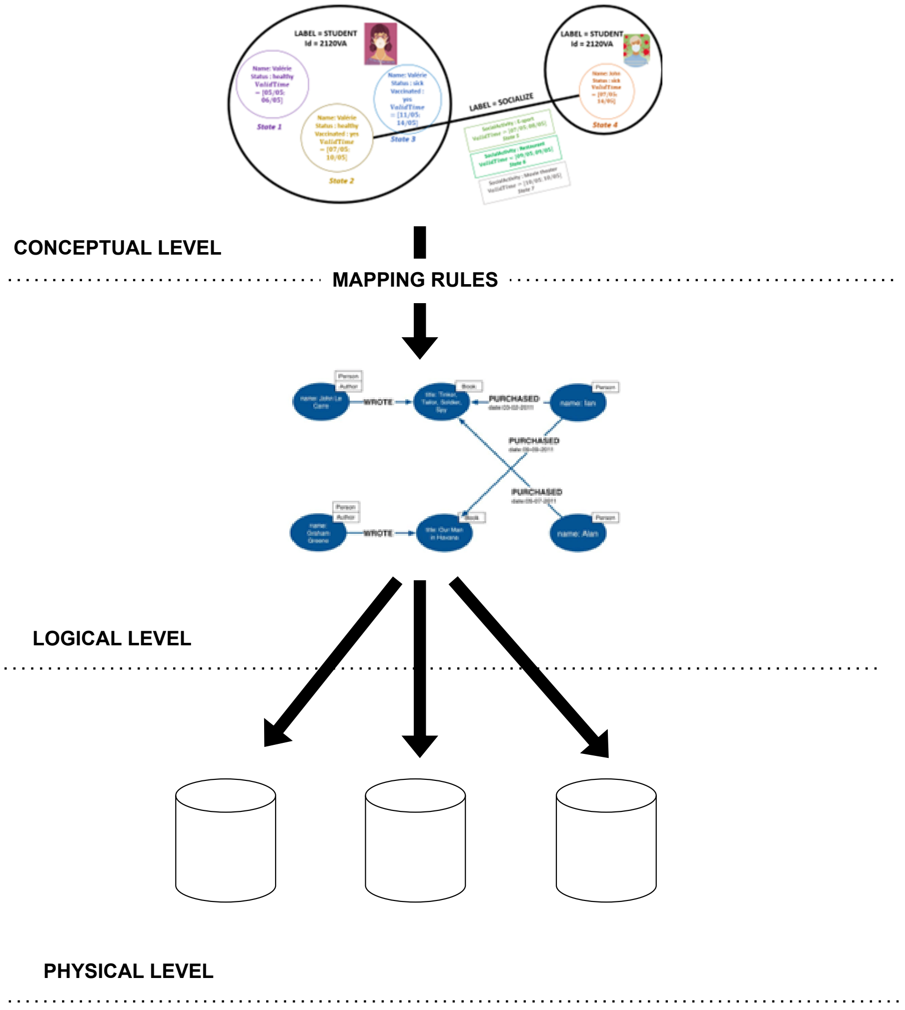


Figure 2.1: Our management solution of TG

2 Preliminary

Before getting to the heart of the matter, we have to define the concepts of graph and temporal graph.

2.1 Graph

The concept of '*Graphs*' originates from the Graph Theory field with Königsberg bridge problem posed by Leonhard Euler in 1735[1]. Graph Theory, a discrete mathematics subbranch, is ultimately the study of connections between things. These things are referred 'vertices' or 'nodes' which are interconnected by 'edges'.

Definition 1. *More formally, a graph is denoted $G = (V, E)$ where V is a set of vertices (or nodes) and E is a set of edges connecting vertices.*

[1] https://interworks.com/blog/nlaurenti/2014/10/20/brief-history-graphs/

The graph is used to model pairwise real-world relationships (with edges) between real-world entities (with vertices) to solve mathematical problems. The Königsberg bridge problem consists of modelling the Königsberg city as a graph (that is, land areas connected by bridges) in order to solve path finding problems across the city. It is only since the 1990s that the concept of graph became popular in the field of data management, due to the lack of hardware support for managing large graphs.

The first line of development of graphs concerns the data models of Graphs. They extend the basic mathematical definition of Graphs: the labelled graph (a label is a characteristic assigned to each vertex and edge), the RDF model (to represent interconnected data of the World Wide Web), the property graph (a directed, labelled, attributed multigraph) and so on (Angles and Gutierrez, 2018).

The second line of development starting from the 2000s encompasses the implementation of graph data in a dedicated technology: graph database management systems to store and access graph data (e.g., Neo4j[2]), graph query languages to query graph data (e.g., Gremlin[3]), graph-processing frameworks to provide analysis of large graph datasets in a distributed environment (e.g., such as GraphX[4]), and so on. These developments have significantly contributed to the field of graph data management, referring to all methods and techniques to model, store, query and analyse graph data, which exist in various applications (i.e., social networks, recommendation systems, transportation systems, etc).

2.2 Temporal Graph

The integration of time dimension in the static graph allows capturing the evolution of its components. In simpler terms, the concept of temporal evolution in a graph refers to the changes that may occur on the different components of the graph over time (Zaki et al., 2016). We therefore break down the concept of temporal evolution in different types, presented in the following definition.

Definition 2. *In a temporal graph, there exist three types of temporal evolution:*

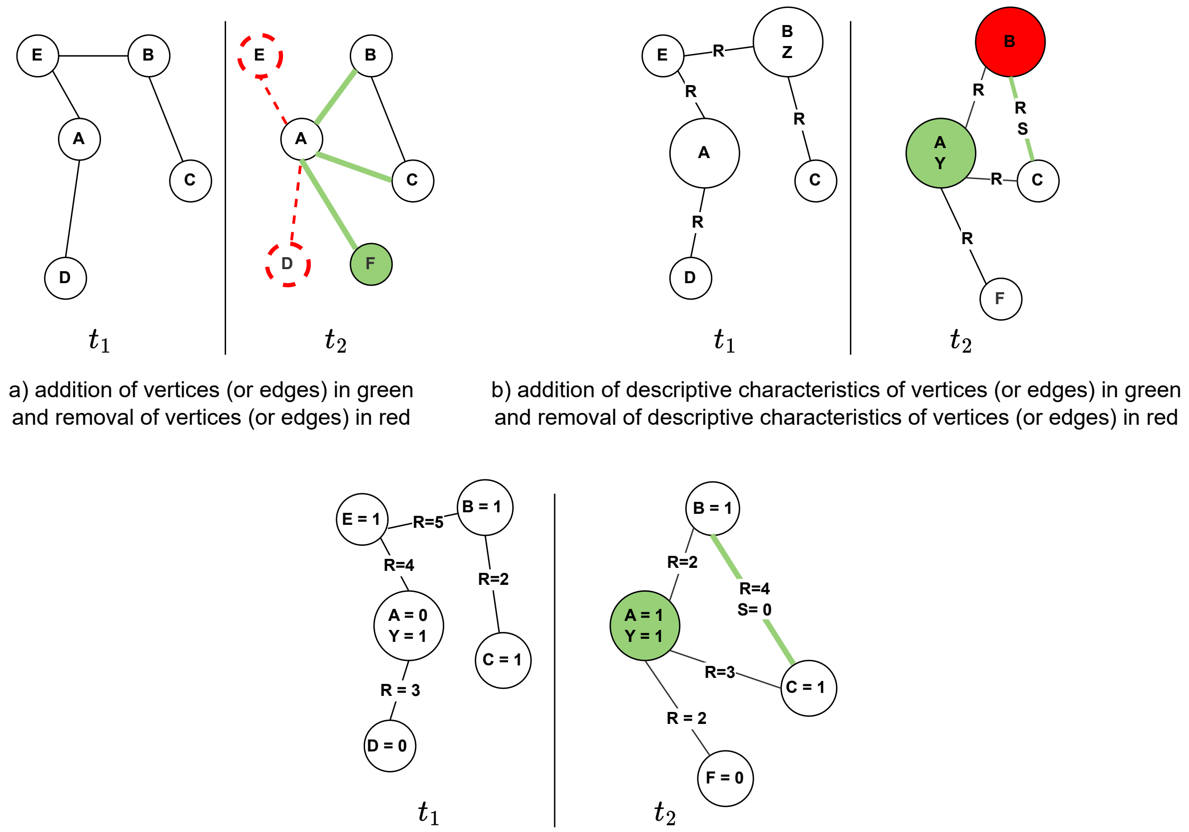
- *the addition and removal of vertices (or edges), we denote as the* ***evolution of topology*** *(Fig 2.2 a));*
- *the addition and removal of vertices' (or edges') descriptive characteristics, we denote as the* ***evolution of attribute set*** *(Fig 2.2 b));*
- *the update of vertices' (or edges') descriptive characteristics' values, we denote as the* ***evolution of attribute value*** *(Fig 2.2 c)).*

The recognition of these changes has led the research community over the last few decades to propose extensions of classic graph models called *'Temporal Graphs'* (Holme and Saramäki, 2012; Debrouvier et al., 2021) and to develop new graph technologies in order to manage graphs with the temporal dimension (Cattuto et al., 2013; Huang et al., 2016).

[2] https://neo4j.com/

[3] https://tinkerpop.apache.org/gremlin.html

[4] https://spark.apache.org/docs/latest/graphx-programming-guide.html



a) addition of vertices (or edges) in green and removal of vertices (or edges) in red

b) addition of descriptive characteristics of vertices (or edges) in green and removal of descriptive characteristics of vertices (or edges) in red

c) update of vertices' (or edges') descriptive characteristics' values in green

Figure 2.2: Evolution in a graph between the time points t_1 and t_2

3 Related Work

In this section, we analyse existing work on management of temporal graph data, i.e., all methods and techniques to model and to implement temporal graph data in a technical environment, at three levels:

- Conceptual level: This level focuses on defining how the concepts of time and evolution are represented and understood within a domain, being the scope of the model;
- Logical level: The previous conceptual level is translated to another data model which is technology-independent, meaning it can be implemented in various technical environments. The objective is to take into account the type of data storage chosen for the implementation;
- Physical Level: The level shifts to the actual implementation of the previous logical model in a chosen technical environment. This involves considerations of optimization or efficiency.

3.1 Conceptual level

At the conceptual level, we distinguish two research axes in the literature: the modelling of time and temporal evolution (Table 2.1).

3.1.1 Time Modelling

Defining the representation of time is a crucial step when modelling temporal graph data. Temporal Graphs are a specialized type of graph models where time is an important aspect. The representation of time in Temporal Graphs is directly inspired by existing work in the field of temporal relational data management, which has been studied extensively since the 1980s (Johnston and Weis, 2010). There are therefore typically two main approaches to represent time in the field of temporal graph modelling: *point-based* and *interval-based* data models (Bohlen et al., 1998). These two approaches directly impact the way the evolution (i.e., changes in the graph) is interpreted as well as the implementation of the data model (at the level of storage and query processing).

Point-based approach The point-based data model is characterized by two features: (i) snapshots of a graph are taken systematically (based on pre-defined rules to capture), with the timestamp of the capture assigned to each snapshot, and (ii) snapshots are taken in a global manner (capturing all data of the graph). In other terms, time is represented through the timestamps at which each snapshot is taken. As a result, we obtain a representation of the states of a graph at successive discrete timestamps (Desmier et al., 2012; Rossi et al., 2013; Yang et al., 2014; Aslay et al., 2018). It is called the 'sequence of graph snapshots' (Debrouvier et al., 2021). It is the classic approach to model temporal graph data. However, the snapshot-based approach has certain drawbacks:

- **Time-window size effects**: The choice of the time-window for constructing the sequence of snapshots can significantly impact the representation of a temporal graph.
  - **Window size selection** : Selecting an appropriate time-window size is not always straightforward. It is a research domain in itself, and the choice can affect the granularity of the temporal information captured (Holme, 2015);
  - **Loss of temporal information** : If the time window is too large, we converge towards a static graph representation where all data is aggregated and all temporal information is then lost (Nicosia et al., 2013);
  - **Data redundancy**: Conversely, if the time-window is too small, the snapshots in the sequence may exhibit similarities between them, and then capturing redundant data (Kosmatopoulos et al., 2016). Besides, at the implementation step, data redundancy can lead to high data volume regarding storage.
- **Indirect access to changes**: Accessing information about changes in a temporal graph involves comparing consecutive snapshots to identify additions, deletions, or modifications of vertices and edges (Moffitt and Stoyanovich, 2017b). Besides, at the implementation step, this process can be computationally intensive and complex, impacting query performance.
- **Lack of finer data granularity**: Snapshots are taken at the level of the entire graph, which means that changes are tracked from a global perspective. This can be a limitation when we need to analyse the evolution of specific vertices or edges (Kosmatopoulos et al., 2016).

Interval-based approach Some works propose the interval-based modelling approach that differ from the point-based modelling approach (Campos et al., 2016; Zhao et al., 2020; Debrouvier et al., 2021). In the interval-based approach, each graph component (i.e. vertex or edge) is associated with a time interval. In other terms, time is represented by time intervals attached to each graph component, indicating the stability period of its state (Jensen et al., 1992). Compared to the point-based approach, the interval-based approach has therefore a finer data granularity which provides the following advantages :

- **Data redundancy reduction**: As each graph component's state is represented within its own time interval, there is a reduction in data redundancy compared to the point-based approach. Besides, at the implementation step, this can lead to more efficient storage, especially when multiple graph components share the same state over extended periods;

- **Direct access to changes**: The interval-based approach allows for direct access to information about changes within the graph. Besides, at the implementation step, The model simplifies queries related to the evolution of the temporal graph, as we can directly access change information within specific intervals;

- **Finer data granularity**: Each graph component can change at its own rate, making it more adaptable to scenarios where different parts of the graph evolve at different speeds. We can therefore track changes from a local perspective (i.e., from a vertex's or edge's viewpoint).

In conclusion, interval-based models are more promising than point-based models. However, the querying aspect must not be neglected (Bohlen et al., 1998).

3.1.2 Temporal Evolution Modelling

The modelling of temporal graph data implies to represent the concept of temporal evolution. As presented in Section 2.2, temporal evolution breaks down into three types of evolution: (i) the evolution of topology, (ii) the evolution of attribute set and, (iii) the evolution of attribute value.

Some temporal graph models represent one above-mentioned evolution type. Yang et al. (2014) focus on the evolution of topology through the addition and removal of edges only. Rossi et al. (2013); Aslay et al. (2018); Latapy et al. (2018) focus also on the evolution of topology through the addition and removal of both vertices and edges.

Some temporal graph models include two evolution types. Desmier et al. (2012) represent the evolution of topology through the addition and removal of edges and the evolution of attribute values through the changes in the value of vertices' attributes. Zhao et al. (2020)'s data model embeds both the evolution of topology through the addition and removal of edges and the evolution of attribute values through the changes in the value of edges' attributes.

Finally, some works try to take into account all evolution types in the proposed temporal graph models (Debrouvier et al., 2021; Campos et al., 2016). However, their models rely on a flat structure. They create new vertices for each descriptive attribute of the vertices of the original graph, and ignore the attributes of edges. So, the changes of attribute set and attribute value are not managed for edges.

Table 2.1: Temporal graph models at the conceptual level. The capital letters in the table are defined as follows: P = Point-based data model, I = Interval-based data model, V = Vertex, E = Edge

Model	**Time Approach**	**Evolution type**		
		Topology	Attribute value	Attribute set
Evolving graph (Yang et al., 2014)	P	E		
Dynamic network (Rossi et al., 2013; Aslay et al., 2018)	P	V/E		
Dynamic attributed graph (Desmier et al., 2012)	P	E	V	
Stream graph (Latapy et al., 2018)	I	V/E		
Attributed Dynamic Graph (Zhao et al., 2020)	I	E	E	
Temporal property graph (Debrouvier et al., 2021; Campos et al., 2016)	I	V/E	V	V
Our model	I	V/E	V/E	V/E

Table 2.2: Implementation of temporal graph models at the physical level

Research axis	**Work**	**Purpose**	**Setting**
Data redundancy reduction	Ren et al. (2011)	Snapshot storage and retrieval	Centralized
	Khurana and Deshpande (2013)	Snapshot storage and retrieval	Distributed
	Khurana and Deshpande (2016)	Historical graph storage and analysis, Node-centric model	Distributed
	Gandhi and Simmhan (2020)	Temporal graph storage and algorithms, Interval-centric model	Distributed
	Ramesh et al. (2020)	Temporal property graph, Interval-centric model, Temporal path queries	Distributed
	Xiangyu et al. (2020)	Snapshot storage and retrieval, Distribution of historical queries	Centralized
Implementation environment	Cattuto et al. (2013)	Modelling, storing and querying time-varying graphs, Neo4j	Centralized
	Huang et al. (2016)	Temporal graph data management system, ACID transactions, Neo4j	Centralized

3.2 Logical level

Traditionally, the translation between the conceptual level and the logical level is framed by rules, such as in the relational data management domain.

In classic graph data management, two common logical data models are mentioned: the property graph model and the RDF (Resource Description Framework) model. They have gained widespread adoption and support within the industry. They are indeed supported by various technical environments (Angles and Gutierrez, 2018).

The property graph model is used for representing entities and relationships. It involves vertices (or nodes) connected by edges (relationships), with attributes stored in both vertices and edges (Angles, 2018). The RDF model focuses on representing semantic links between data using vertices (resources) linked by edges (triples). It does not allow nesting data (i.e., attributes) within vertices and edges, but is highly suitable for semantic data representation (Lassila and Swick, 1998).

3.3 Physical level

At the physical level, we distinguish two research axes in the literature: data redundancy reduction and implementation environment (Table 2.2).

Regarding the first research axis, snapshots inevitably introduce data redundancy since consecutive snapshots share in common vertices and edges that do not change over time Kosmatopoulos et al. (2016). Processing snapshots causes redundant computation, limiting scalability Gandhi and Simmhan (2020). In response to this issue, Ren et al. (2011) propose a framework to construct a few representative graphs based on similarity. Khurana and Deshpande (2013) introduce an in-memory data structure and a hierarchical index structure to retrieve efficiently snapshots of a temporal graph. Xiangyu et al. (2020) proposes a strategy to determine when snapshots should be materialized based on the distribution of historical queries. However, these optimization techniques snapshots always accept some data redundancy. To avoid data redundancy, some works recommend using interval-based data model completely in break with snapshots. However, they are oriented towards distributed computing, so do not provide a conceptual-oriented view (Khurana and Deshpande, 2016; Gandhi and Simmhan, 2020; Ramesh et al., 2020).

Regarding the second research axis, some works focus on evaluating the performance of graph data management systems supporting temporal graph models via experimental assessments. Some experiments rely on RDF triple stores, such as Virtuoso[5] or TDB-Jena[6], to store the evolution of Linked Open Data (LOD) in the Semantic Web area (Roussakis et al., 2015; Pernelle et al., 2016). However, it is already known that graph oriented NoSQL databases are more efficient than RDF triple stores when querying RDF data (Ravat et al., 2019). It is necessary to see if these NoSQL databases are as efficient in the context of temporal graphs. The authors in Cattuto et al. (2013) use Neo4j to store the time-varying networks and to retrieve specific snapshots. The authors in Huang et al. (2016) have developed a graph database management system based on Neo4j to support graphs changing in the value of vertices' and edges' attributes, but do not address the changes in graph topology.

[5] `https://virtuoso.openlinksw.com/`

[6] `https://jena.apache.org/documentation/tdb/`

3.4 Summary

We observe in the previous related work that they hardly separate the modelling and implementation details of temporal graph data. They therefore lack of comprehensive overview of graph data and their temporal evolution and reproducibility in various contexts. To solve this problem, we will propose a complete solution divided in three abstraction levels: conceptual, logical and physical.

At the conceptual level, we propose a data model using the interval-based approach for time modelling, since it presents several advantages over the point-based approach. Moreover, it includes all possible evolution types of graph data (topology, attribute set and attribute value) to be generic in terms of evolution representation.

At the logical level, we propose direct mapping rules to translate our conceptual model into the property graph model. It allows benefiting from established technologies and practices that are compatible with the property graph model. Introducing a new, proprietary logical model from scratch can lead to compatibility challenges with existing technologies.

At the physical level, we propose to implement the property graph model into a graph-oriented NoSQL data store and to evaluate its performance in terms of storage and querying.

4 Conceptual Modelling

In this chapter, we define a conceptual modelling of temporal graph data. First, we define the concept of time in our model (Section 4.1). Second, we define our temporal graph (Section 4.2). Third, we describe the evolution mechanism in our proposed concepts (Section 4.3). Finally, we illustrate the proposed concepts in an example (Section 4.4).

4.1 Time

Time can be schematized as a domain denoted by Ω, which is linear and discretized by ordered natural numbers corresponding to their succession in time (Gandhi and Simmhan, 2020; Ramesh et al., 2020). Each time point corresponds to an *instant*.

Definition 3. *A time unit is an atomic increment in time defined by some user (Gandhi and Simmhan, 2020; Ramesh et al., 2020). It is defined by a mapping function $T(x) \subset 2^{\mathbb{N}}$. $T(x)$ allows associating a time interval, indexed by $x \in \mathbb{N}$, to a set of instants (Fig 2.3). A time unit has therefore the following characteristics (Wang et al., 1993):*

- $0 \in T(0)$ $\{$*each time unit starts from the beginning*$\}$;
- $\forall i, j \in \mathbb{N}, i \neq j \rightarrow T(i) \cap T(j) = \emptyset$ $\{$*two continuous blocks do not overlap*$\}$;
- $\forall i \in \mathbb{N}, \exists j \in \mathbb{N}$ *such that* $i \in T(j)$ $\{$*each time unit covers the whole timeline, i.e.,* $\mathbb{N}\}$.

The most common units are corresponding to the usual partitions of calendars are: millennium, century, year, month, day, week, hour, second, etc. A time unit can be the partition of another such as days for months.

Definition 4. *A time interval defines a set of instants between two instant limits in time. We denote it $T = [t_b, t_f]$ where, $t_b, t_f \in \Omega$ which indicates a time interval beginning at t_b and extending to t_f. The duration of the time interval is $t_f - t_b$. A time instant is therefore a time interval $T = [t_b, t_f]$ where $t_b = t_f$ and $t_b, t_f \in \Omega$. It has no duration relatively to its time unit.*

In our model, we use the concept of *the valid time interval.*

Definition 5. *A valid time interval is a time interval, $T = [t_b, t_f]$, which is associated to a graph component indicating its stability period (i.e., the period during which it does not change) (Jensen et al., 1992).*

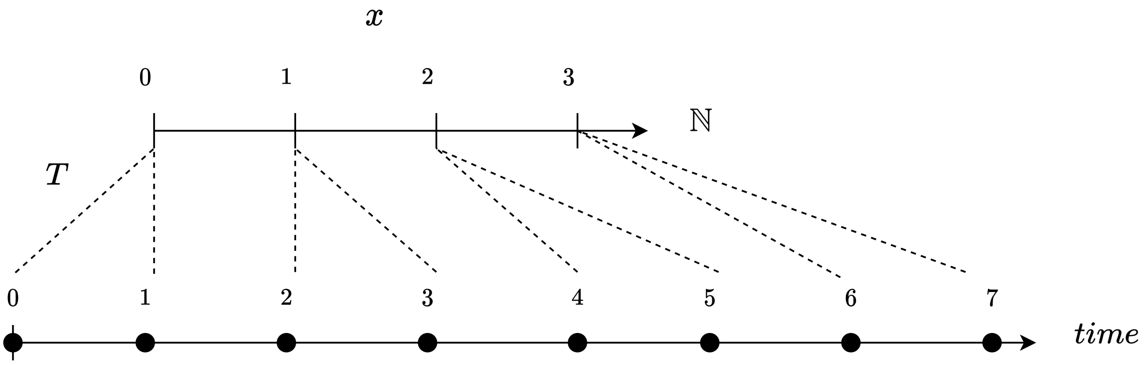


Figure 2.3: Time modelling.

4.2 Temporal Graph

We define a conceptual model, named *Temporal Graph*, for representing graph data that changes over time. More specifically, we propose concepts to model entities of a business context that can change over time, as well as the relationships between these entities.

We describe an entity of a business context that changes over time with the concept of *temporal entity.* A temporal entity is composed of several *states* to represent its evolution over time.

Definition 6. *A temporal entity, called e_i, is defined by $\langle l^{e_i}, id^{e_i}, S^{e_i}, T^{e_i}\rangle$ where l^{e_i} is the label of e_i, id^{e_i} is the identifier of e_i, $S^{e_i} = \{s_1^{e_i}, ..., s_m^{e_i}\}$ is the non-empty set of states of e_i and T^{e_i} is the valid time interval of e_i. Each state $s_j^{e_i} \in S^{e_i}$ is defined by $s_j^{e_i} = \langle A^{s_j}, V^{s_j}, T^{s_j}\rangle$ where $A^{s_j} = \{a_1^{e_i}; ...; a_n^{e_i}\}$ is the set of attributes of $s_j^{e_i}$, $V^{s_j} = \{v(a_1^{e_i}); ...; v(a_n^{e_i})\}$ is the attribute values and T^{s_j} is the valid time interval of $s_j^{e_i}$. Each $v(a_q^{e_i}) \in V^{s_j}$ is the value of each attribute $a_q^{e_i} \in A^{s_j}$.*

Definition 7. *The valid time interval of each state of a temporal entity $s_j^{e_i} \in S^{e_i}$ is defined by $T^{s_j} = [t_b, t_f]$ where $t_b \neq \emptyset$ and $t_f \neq \emptyset$. The valid time interval of each temporal entity e_i is obtained by calculation:*

$$T^{e_i} = \cup_{j=1}^{j=m} T^{s_j} \text{ where } s_j \in S^{e_i}$$

A relationship between two entities of a business context does not have an independent existence. Its existence depends on the entities it links. We describe a relationship between two entities that changes over time with the concept of *temporal relationship*. A temporal relationship is composed by several *states* to represent its evolution over time.

Definition 8. *A temporal relationship, called r_i, is defined by $\langle l^{r_i}, (s_k, s_j), S^{r_i}, T^{r_i} \rangle$ where l^{r_i} is the label of r_i, (s_k, s_j) is the couple of entity states r_i links, $S^{r_i} = \{s_1^{r_i}, ..., s_u^{r_i}\}$ is the non-empty set of states of r_i and T^{r_i} is the valid time interval of r_i. Each state $s_b^{r_i} \in S^{r_i}$ is defined by $s_b^{r_i} = \langle A^{s_b}, V^{s_b}, T^{s_b} \rangle$ where $A^{s_b} = \{a_1^{r_i}; ...; a_w^{r_i}\}$ is the set of attributes of $s_b^{r_i}$, $V^{s_b} = \{v(a_1^{r_i}); ...; v(a_w^{r_i})\}$ is the attribute values and T^{s_b} is the valid time interval of $s_b^{r_i}$. Each $v(a_d^{r_i}) \in V^{s_b}$ is the value of each attribute $a_d^{r_i} \in A^{s_b}$.*

Remark 1. *The valid time interval of each state of a temporal relationship $s_b^{r_i} \in S^{r_i}$ is defined by $T^{s_b} \subseteq (T^{s_k} \cap T^{s_j})$ where T^{s_k} is the valid time of the entity state s_k and T^{s_j} is the valid time of the entity state s_j. The valid time interval of each temporal relationship r_i is obtained by calculation:*

$$T^{r_i} = \cup_{b=1}^{b=u} T^{s_b} \text{ where } s_b \in S^{r_i}$$

Definition 9. *L describes a finite set of labels. A label $l \in L$ describes the semantic of entities (or relationships). Therefore, their definition is domain-specific. A label groups an entity class (or relationship class). Conversely, an entity (or relationship) has a unique label. Unlabelled entities (or relationships) are semantically indistinct.*

As a result of the previous definitions, our *Temporal Graph* is defined as follows:

Definition 10. *A Temporal Graph, called TG, is defined by $TG = \langle E, R, T, \rho, \lambda \rangle$ where:*

- *$E = \{e_1, ..., e_g\}$ is a finite set of temporal entities;*
- *$R = \{r_1, ..., r_h\}$ is a finite set of temporal relationships;*
- *T is the timeline of the temporal graph. It only depends on the valid time intervals of temporal entities, as they have an independent existence. So it is obtained by calculation:*

$$T = \cup_{i=1}^{i=g} T^{e_i} \text{ where } e_i \in E \tag{2.1}$$

- *$\rho : R \rightarrow (E \times E)$ is a function that associates each state of each relationship in R with a pair of entity states in E;*
- *$\lambda : (E \cup R) \rightarrow SET^+(L)$ [7] is a function that associates each entity (or relationship) in the temporal graph with a label from L.*

Definition 11. *The schema of the Temporal Graph is a tuple $schema(TG) = (L^E, L^R, \phi)$ where:*

- *$L^E \subset L$ is a finite set of labels;*
- *$L^R \subset L$ is a finite set of labels, satisfying that L^E and L^R are disjoint;*

[7] SET^+ denotes a non-empty set

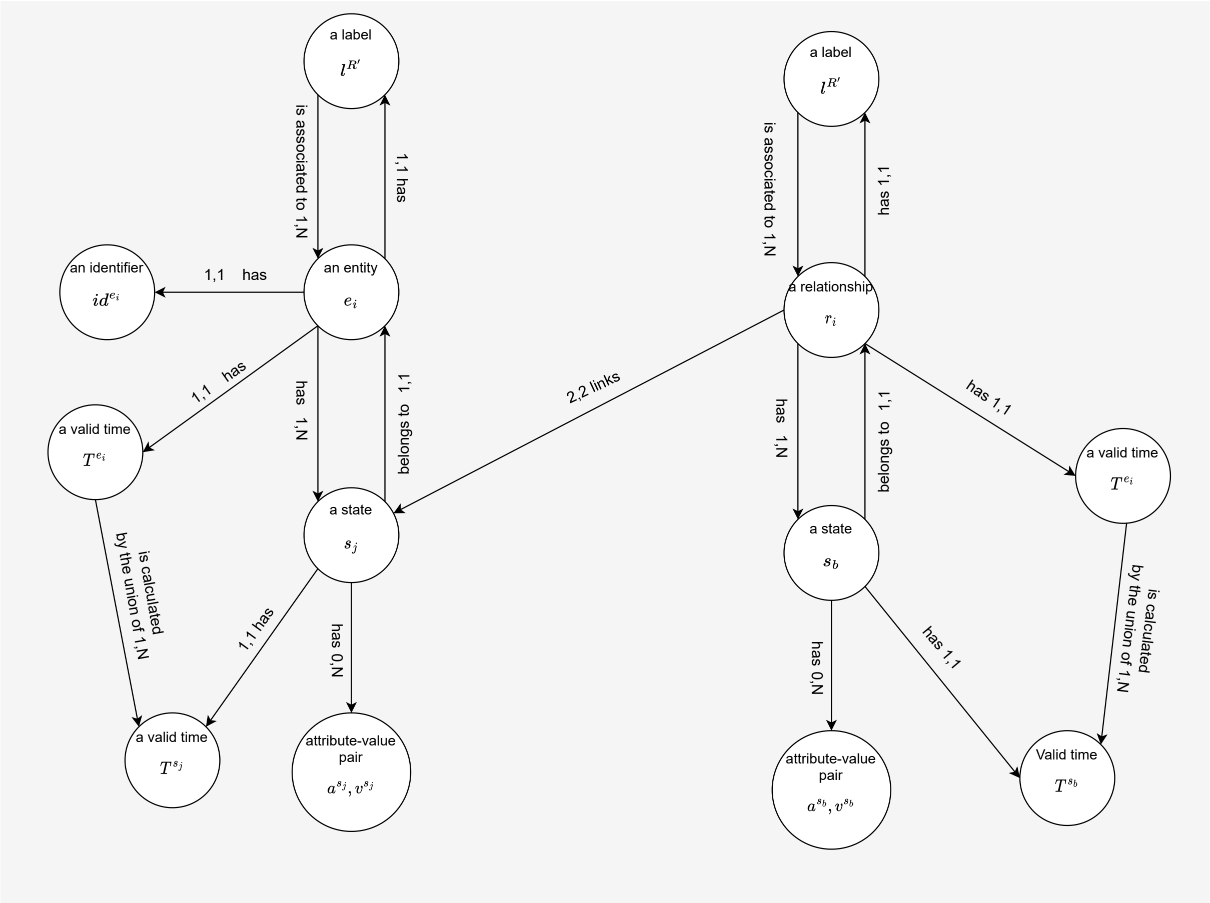


Figure 2.4: Metamodel of our conceptual modelling

- $\phi : (L^E, L^E) \rightarrow SET^+(L^R)$ *is a function that defines the finite and non-empty subset of relationship labels from* L^R *allowed between a given pair of entity labels.*

The metamodel in Fig 2.4 presents the relationships (edges) between all the concepts (vertices) of our conceptual modelling by specifying the cardinalities of these relationships. We present the graphical notation of our conceptual modelling in Fig 2.7 in the following section.

4.3 Temporal Evolution

As seen in the conceptual modelling above, we describe the temporal evolution of entities and their relationships though the concept of *states*. In this section, we precise how we manage the three types of evolution: (i) *the evolution in topology*, (ii) *the evolution in attribute set* and (iii) *the evolution in attribute value*.

The management of the topology evolution of entities is presented in Table 2.3. When an entity e_i is added in the modelled business context, a state of the entity s_j is created, and its valid time interval is $T^{e_i} = T^{s_j} = [t_{addition}, +\infty)$ where $t_{addition}$ is the time instant of the addition. When an entity e_i is removed from the modelled business context, the valid time interval of its last state is updated to $T^{s_m} = [t_b, t_{removal}[$ with $t_{removal}$ the time instant of the removal and $t_b < t_{removal}$. Similarly, the valid time interval of e_i is updated to $T^{e_i} = T^{e_i} \cup \{T^{s_m}\}$.

Table 2.3: Evolution of topology in TG

Input:	$TG = \langle E, R, T \rangle$
Output:	$TG = \langle E', R', T' \rangle$
Actions:	1. If an entity e_i appears at $t_{addition}$ Then 2. $\quad s_m = CreateState(A^{s_j}, V^{s_j}, [t_{addition}, +\infty))$ 3. $\quad S^{e_i} \leftarrow s_m$ 4. $\quad e_i = CreateEntity(l^{e_i}, id^{e_i}, S^{e_i}, T^{e_i})$ 5. $\quad E \leftarrow E \cup \{e_i\}$ 6. Else If $e_i \in E$ is removed at $t_{removal}$ Then 7. $\quad s_m = GetLastState(e_i)$ 8. $\quad T^{s_m} = UpdateEndTime(t_{removal})$ 9. End If 10. If a relationship r_i appears at $t_{addition}$ Then 11. $\quad s_m = CreateState(A^{s_j}, V^{s_j}, [t_{addition}, +\infty))$ 12. $\quad S^{r_i} \leftarrow s_m$ 13. $\quad r_i = CreateRelationship(l^{r_i}, (s_k, s_j), S^{r_i}, T^{r_i})$ 14. $\quad R \leftarrow R \cup \{r_i\}$ 15. Else If $r_i \in R$ is removed at $t_{removal}$ Then 16. $\quad s_m = GetLastState(r_i)$ 17. $\quad T^{s_m} = UpdateEndTime(t_{removal})$ 18. End If

The management of the evolution in attribute set and attribute value of entities is presented in Table 2.4. A temporal entity changes according to the addition and/or removal of a new attribute and/or the update in an attribute value. Two states of the same entity have different attribute sets and/or different attribute values. When a new attribute is added/removed or an attribute value changes, a new state of the entity is created instead of overwriting the old state version. The valid time interval of the old state version, s_j, is updated to $T^{s_j} = [t_b, t_{change}[$ where t_{change} is the time instant of the change and $t_b < t_{change}$. The valid time interval of the new state version, s_{j+1}, is $T^{s_{j+1}} = [t_{change}, +\infty)$.

Similar to entities, relationships can evolve in terms of topology, attribute value or attribute set. So to capture the evolution of temporal relationships, we apply the same evolution management.

4.4 Example

The difficulty in exploiting a dataset with temporal graph data is to follow how data relates to each other and how data changes over time. To do so, such dataset can be ideally represented as a temporal graph using our conceptual model. In the following, we present the modelling of a dataset of an e-commerce activity into our temporal graph representation and present its advantage compared to a snapshot-based representation.

In our business use case, customers view, add to cart and buy items on an e-commerce website. They can make a new action (i.e. view, add to cart and buy) on items each minute. They can modify characteristics of their cart over time by changing items' quantity or by adding a discount code. The website adds new items over time. Moreover, it

Table 2.4: Evolution of attribute set and attribute value in TG

Input:	$TG = \langle E, R, T \rangle$
Output:	$TG = \langle E', R', T' \rangle$
Actions:	1. If a new attribute a_i of e_i is added at t_{change} Then 2. $s_j = GetLastState(e_i)$ 3. $T^{s_j} = UpdateEndTime(t_{change})$ 4. $s_{j+1} = CreateState(A^{s_j} \cup \{a_i\}, V^{s_j} \cup \{v(a_i)\}, [t_{change}, +\infty))$ 5. Else If an attribute a_i of e_i is removed at t_{change} Then 6. $s_j = GetLastState(e_i)$ 7. $T^{s_j} = UpdateEndTime(t_{change})$ 8. $s_{j+1} = CreateState(A^{s_j} \setminus \{a_i\}, V^{s_j} \setminus \{v(a_i)\}, [t_{change}, +\infty))$ 9. Else If the attribute value $v(a_i)$ of e_i changes to $v'(a_i)$ at t_{change} Then 10. $s_j = GetLastState(e_i)$ 11. $T^{s_j} = UpdateEndTime(t_{change})$ 12. $V^{s_{j+1}} \leftarrow V^{s_j} \setminus \{v(a_i)\} \cup \{v'(a_i)\}$ 13. $s_{j+1} = CreateState(A^{s_j}, V^{s_{j+1}}, [t_{change}, +\infty))$ 14. End If 15. If a new attribute a_i of r_i is added at t_{change} Then 16. $s_j = GetLastState(r_i)$ 17. $T^{s_j} = UpdateEndTime(t_{change})$ 18. $s_{j+1} = CreateState(A^{s_j} \cup \{a_i\}, V^{s_j} \cup \{v(a_i)\}, [t_{change}, +\infty))$ 19. Else If an attribute a_i of r_i is removed at t_{change} Then 20. $s_j = GetLastState(r_i)$ 21. $T^{s_j} = UpdateEndTime(t_{change})$ 22. $s_{j+1} = CreateState(A^{s_j} \setminus \{a_i\}, V^{s_j} \setminus \{v(a_i)\}, [t_{change}, +\infty))$ 23. Else If the attribute value $v(a_i)$ of r_i changes to $v'(a_i)$ at t_{change} Then 24. $s_j = GetLastState(r_i)$ 25. $T^{s_j} = UpdateEndTime(t_{change})$ 26. $V^{s_{j+1}} \leftarrow V^{s_j} \setminus \{v(a_i)\} \cup \{v'(a_i)\}$ 27. $s_{j+1} = CreateState(A^{s_j}, V^{s_{j+1}}, [t_{change}, +\infty))$

adds new characteristics to items and updates the value of characteristics over time.

To model such e-commerce data, we propose a two-step approach :

- The first step consists of identifying the entities and relationships that model the business needs. In our conceptual model, each entity and relationship classes of a business domain are modelled through the concept of labels.
- The second step consists of identifying the evolution of the various components of the previous schema. In our conceptual model, each entity of a business domain that evolves over time is modelled through the concept of temporal entity. Each relationship of a business domain that evolves over time is modelled through the concept of temporal relationship. All descriptive information of entities and relationships of a business domain are modelled through the concept of attributes. We manage their evolution, in terms of topology - attribute set - attribute value, notably through the

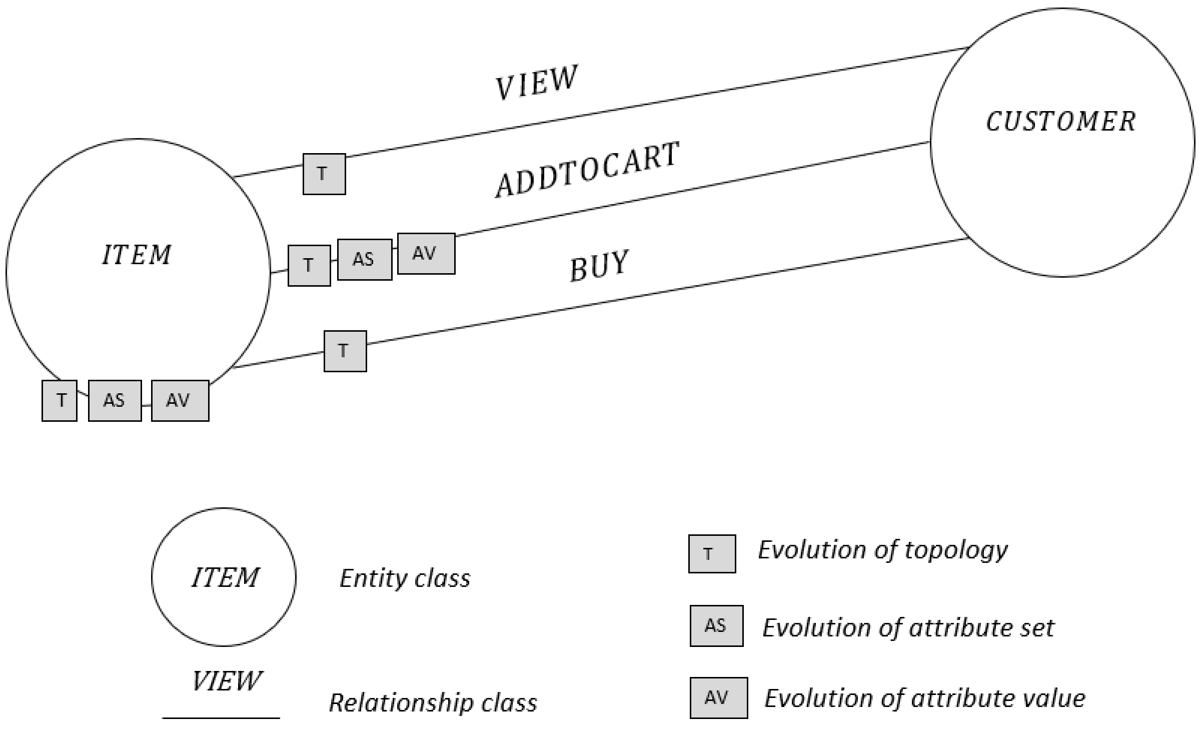


Figure 2.5: Schema of the e-commerce dataset

concept of states.

In the first step, we identify two entity classes (customer and item) and three relationship classes (view, add to cart and buy). So, the formal description of the temporal graph schema is $schema(TG) = (L^E, L^R, \phi(L^E, L^E)$ where:

- $L^E = \{CUSTOMER, ITEM\}$
- $L^R = \{VIEW, ADDTOCART, BUY\}$
- $\phi(CUSTOMER, ITEM) = \{VIEW, ADDTOCART, BUY\}$

We graphically construct the schema of the dataset in Fig 2.5.

In the second step, we understand that customers do not change over time, contrary to items. Items evolve over time in terms of their topology, attribute set and attribute value. Similarly, the relationships 'add to cart' evolve over time in terms of their topology, attribute set and attribute value. The relationships 'view' and 'buy' evolve over time in terms of their topology only. We specify these evolution types on the schema of the dataset in Fig 2.5. According to our conceptual modelling, customers and items become temporal entities. Actions of customers on items (view, add to cart and buy) become temporal relationships. The characteristics of customer, items and carts are translated into attributes. We illustrate in details the modelling of the evolution of these entities and relationships through the following business scenarios in the dataset.

A customer identified $C1$ and called 'Smith' never experiences a change in its characteristics since the creation of its account. If we use the snapshot-based approach, we consider that data are captured at a regular time interval, for instance each day. There-

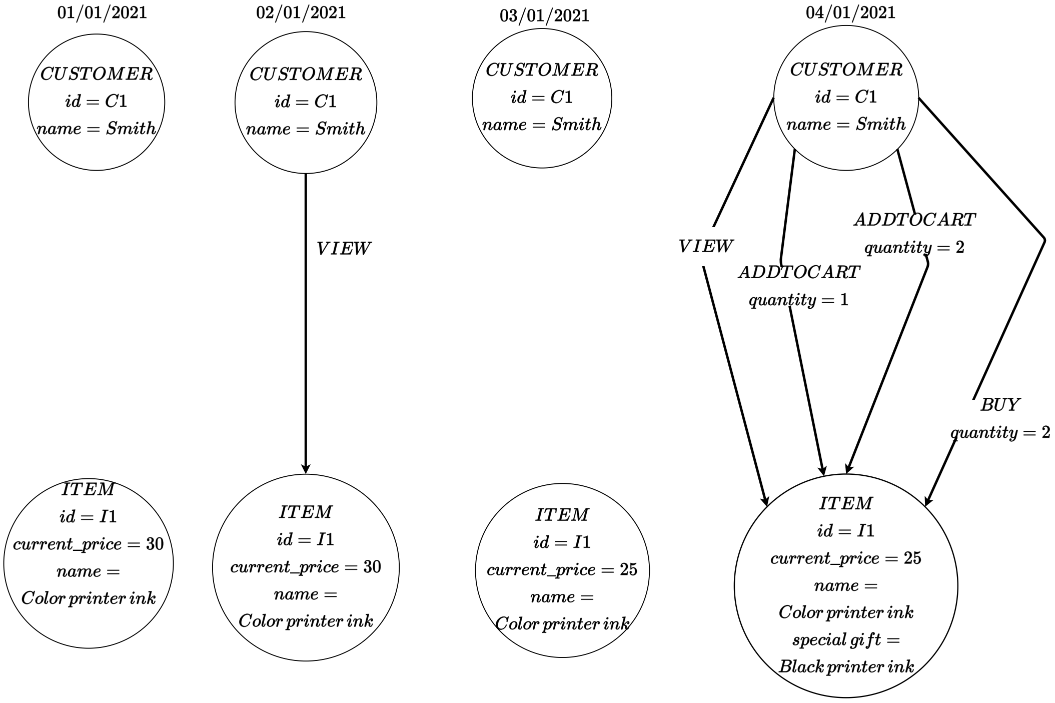


Figure 2.6: Evolution management of the e-commerce dataset with the snapshot-based model

fore, the vertex representing the customer is repeated at each snapshot, as we can see in Fig 2.6. The advantage of our conceptual model is to represent this customer by only one state, numbered 1 in Fig 2.7, with a start valid date corresponding to the creation date of its account and no ending date. The formal description of this customer according to our conceptual model is given as follows :

- $e_1 = \langle CUSTOMER, C1, \{s_1\}, [01/01/2021, +\infty) \rangle$
- $s_1 = \langle \{name\}, \{Smith\}, [01/01/2021, +\infty) \rangle$

The website adds new items over time. This concerns notably the item 'Color printer ink' identified as $I1$. At its publication on the website, the price of $I1$ is 30. Two days after $I1$'s publication, its *price* has decreased. This refers to the evolution in attribute value of $I1$. The next day, the website has added a new descriptive information (*special gift*) to $I1$. This refers to the evolution in attribute set of $I1$. With the snapshot-based approach in Fig 2.6, as $I1$ does not change during two days, the initial state of $I1$ is repeated in two snapshots. In our model, a state is generated at each change. The publication of $I1$ generates the first state of $I1$ numbered 2 in Figure 2.7, with a valid time interval beginning at the date of its publication. The change in $I1$'s attribute value results in the new state, numbered 3, with a valid time interval beginning at the date of the price decrease. The change in $I1$'s attribute set results in the new state numbered 4, with a valid time interval beginning at the date of the attribute addition. Therefore, this produces only 3 vertices for $I1$ in our model instead of 4 vertices in the snapshot-based approach. The formal description of this item according to our conceptual model is given as follows:

- $e_2 = \langle ITEM, I1, \{s_2, s_3, s_4\}, \{[01/01/2021, 02/01/2021], [03/01/2021, 03/01/2021],$

$[04/01/2021, +\infty)\})\rangle$

- $s_2 = \langle \{current\_price, name\}, \{30, Color\,printer\,ink\}, [01/01/2021, 02/01/2021]\rangle$
- $s_3 = \langle \{current\_price, name\}, \{25, Color\,printer\,ink\}, [03/01/2021, 03/01/2021]\rangle$
- $s_4 = \langle \{current\_price, name, special\,gift\}, \{25, Color\,printer\,ink, Black\,printer\,ink\}, [04/01/2021, +\infty)\rangle$

Customers can make a new action on items each minute. This refers to the evolution in topology of relationships between customers and items. The customer $C1$ viewed $I1$ once during the day 02/01/2021 at 10:30. During the day 04/01/2021, the customer $C1$ viewed $I1$ at 10:30, added it to cart at 10:33, modified its cart at 10:37 and then bought it at 10:40. As we can see in Fig 2.6, a snapshot-based approach would lose the temporal information because it does not capture the order of actions or their timestamps. This is because time granularity is at the level of the entire graph and not each graph component (vertex or edge). On the contrary, our model keeps all temporal information since time granularity is at the level of each graph component. So the two actions $VIEW$ and $ADDTOCART$ are represented by temporal relationships with at least one state in Fig 2.7.

The characteristics of customers' actions can be updated. During the day 04/01/2021, the customer $C1$ has modified the quantity of the item $I1$ in his cart following a *discount code* he received from the website. He has added his *discount code* to his cart. This refers respectively to the evolution in attribute value and set of the relationship $ADDTOCART$ between $C1$ and $I1$. As said previously, the order of the information is lost in the snapshot-based approach. In our model, the change in the attribute set and value of the relationship $ADDTOCART$ between $C1$ and $I1$ generates two states in Fig 2.7: (i) a state numbered 7 which is the initial state of the relationship before changes and (ii) a state numbered 8 with one more unit of $I1$'s quantity and a *discount code*. Then, the customer $C1$ bought the item $I1$. This generates the state numbered 9. The formal description of customers' actions is as follows:

- $r_1 = \langle VIEW, (s_1, s_2), \{s_5\}, [02/01/2021\,10:30, 02/01/2021\,10:30]\rangle$
- $r_2 = \langle VIEW, (s_1, s_4), \{s_6\}, [04/01/2021\,10:30, 04/01/2021\,10:30]\rangle$
- $r_3 = \langle ADDTOCART, (s_1, s_4), \{s_7, s_8\}, \{[04/01/2021\,10:33, 04/01/2021\,10:33], [04/01/2021\,10:37, 04/01/2021\,10:37]\}\rangle$
- $r_4 = \langle BUY, (s_1, s_4), \{s_9\}, [04/01/2021\,10:40, 04/01/2021\,10:40]\rangle$
- $s_5 = \langle \emptyset, \emptyset, [02/01/2021\,10:30, 02/01/2021\,10:30]\rangle$
- $s_6 = \langle \emptyset, \emptyset, [04/01/2021\,10:30, 04/01/2021\,10:30]\rangle$
- $s_7 = \langle \{quantity\}, \{1\}, [04/01/2021\,10:33, 04/01/2021\,10:33]\rangle$
- $s_8 = \langle \{quantity, discount\,code\}, \{2, Summer\}, [04/01/2021\,10:37, 04/01/2021\,10:37]\rangle$
- $s_9 = \langle \{quantity\}, \{2\}, [04/01/2021\,10:40, 04/01/2021\,10:40]\rangle$

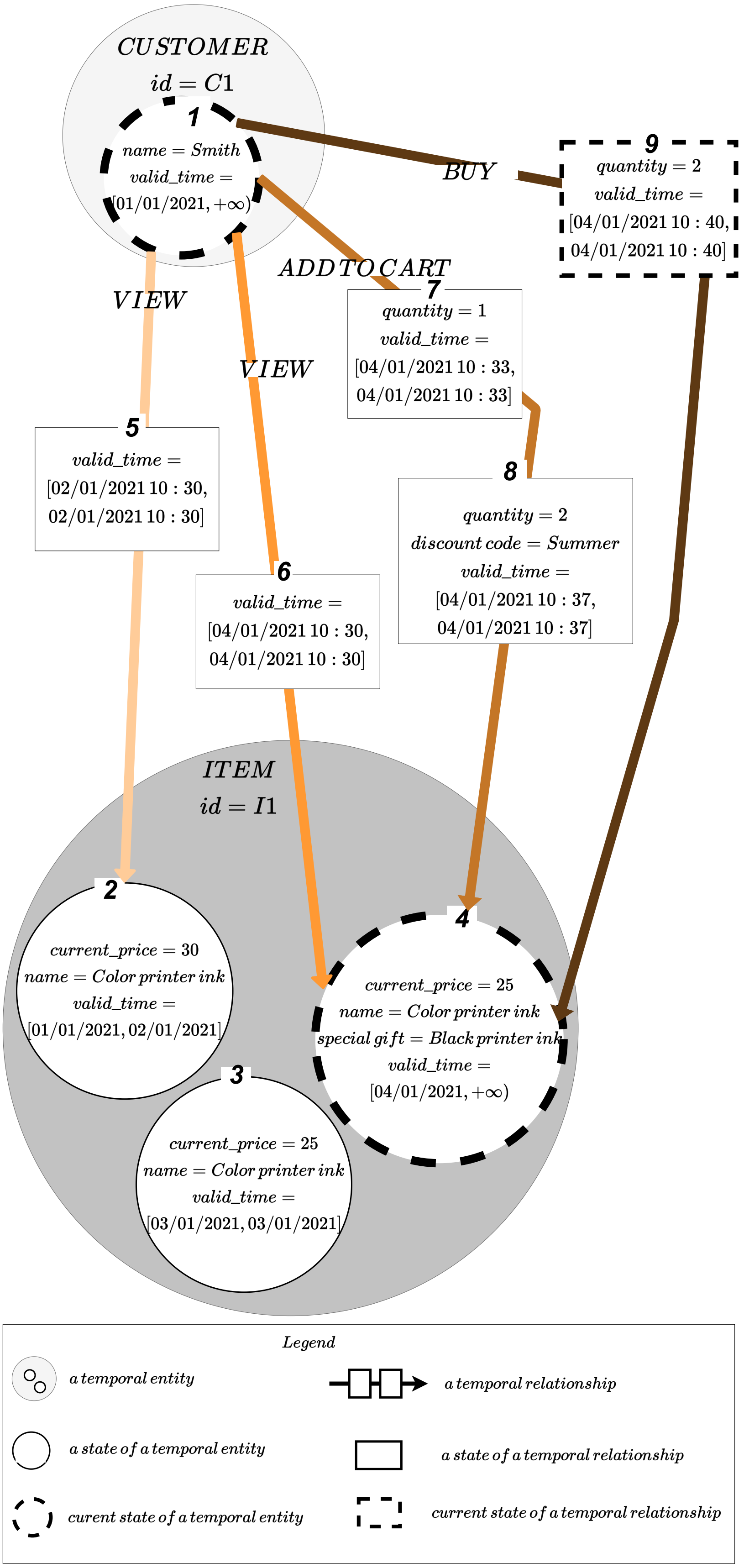


Figure 2.7: Evolution management of the e-commerce dataset with our temporal graph model

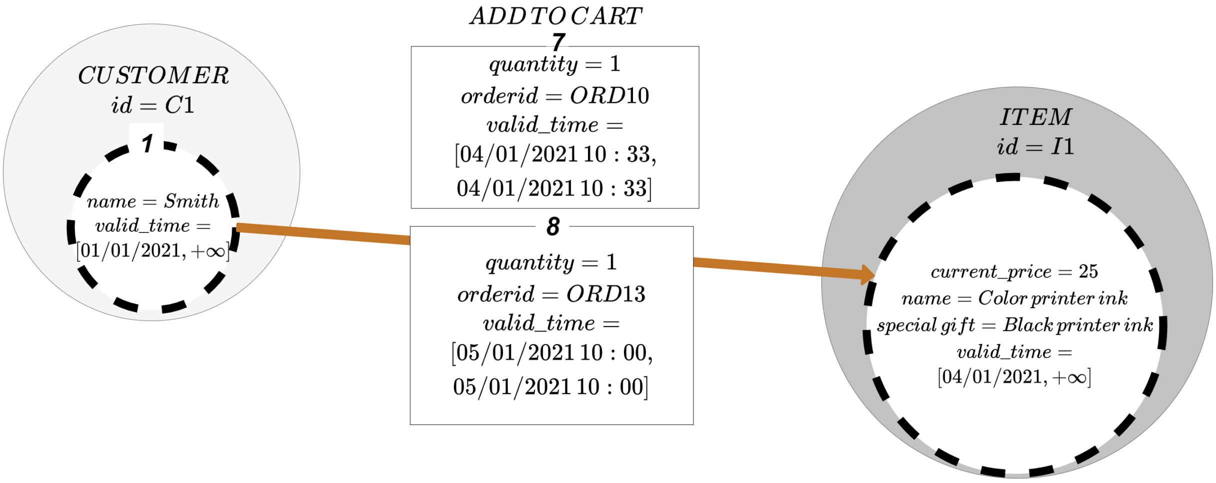


Figure 2.8: Extended example

Through this example, the following advantages of our conceptual model are retained. First, our conceptual model provides a comprehensive overview of graph data with temporal evolution to users (decision makers, analysts, etc.). It captures, in a finer way, the different types of evolution of entities and relationships (topology, attribute set and attribute value). Moreover, it provides a graphical notation that allows for easily representing the topology of data (vertices connected by edges), the data embedded within the topology (attributes) and their temporal evolution (states).

Second, our conceptual model represents the temporal evolution of graph data in a synthetic manner. No information is lost or redundant after the modelling process. If we had adopted the snapshot-based approach, we would have 4 more vertices than our model.

Finally, our conceptual model is flexible in the way of modelling business requirements. Suppose now that the dataset includes a customer that adds to cart the same item (same state of item) at two different dates for two different orders. We can easily do that by adding an attribute ‘order’ for each state of *ADDTOCART* relationship between the same customer (same state of customer) and the same item (same state of item). We illustrate this case in Fig 2.8.

5 Logical modelling

The objective of logical modelling is to take into account the type of data storage chosen for the implementation. In our case, we choose the logical property graph model because most of graph-oriented NoSQL data stores, such as Neo4j, are designed to store property graphs. Our objective is then to translate our conceptual temporal graph into a logical property graph.

According to Angles (2018), a property graph is defined as $PG = \langle N, D, \eta, \Lambda, \Sigma \rangle$ where N is a finite set of nodes (also called vertices), D is a finite set of edges, $\eta : D \to (N \times N)$ is a function that associates each edge in D with a pair of nodes in N, $\Lambda : (N \cup D) \to SET^{+}(L)$ is a function that associates a node (or an edge) with a set of labels from L, and $\Sigma : (N \cup D) \times P \to SET^{+}(V)$ is a function that associates nodes (or edges) with

Conceptual Temporal graph	**Logical Property graph**
a temporal entity e_i	a set of nodes
a label of a temporal entity l^{e_i}	a label
a temporal entity's identifier id^{e_i}	a property
a valid time interval of a temporal entity T^{e_i}	by query
a state of a temporal entity s_j	a node
an attribute of a temporal entity $a_q^{e_i}$	a property
a valid time interval of an entity state T^{s_j}	two properties*
a temporal relationship r_i	a set of edges connecting nodes
a label of a temporal relationship l^{r_i}	a label
the couple of entity states (s_k, s_j) that links a temporal relationship	two nodes
a valid time interval of a temporal relationship T^{r_i}	by query
a state of a temporal relationship s_b	an edge
an attribute of a temporal relationship $a_d^{r_i}$	a property
a valid time interval of a relationship state T^{s_b}	two properties*

Table 2.5: Mapping rules of the conceptual temporal graph into the logical property graph. **startvalidtime* and *endvalidtime*.

properties. Each property is a key-value pair (p, v) where p is the property name and v the property value.

We propose a transformation process between our conceptual temporal graph and a logical property graph via a generic algorithm (Algorithm 1). The transformation process receives our temporal graph TG as input and returns the property graph PG. For each state s of each temporal entity e in G, a node is created in PG with a label corresponding to the label of e and a set of properties corresponding to: the identifier of e, the attributes of s, the start and end instants of the valid time interval of s. For each state s of each temporal relationship r in TG, an edge is created in PG by connecting the two nodes corresponding to two states that r links, with a label corresponding to the label of r and a set of properties corresponding to: the attributes of s, the start and end instants of the valid time interval of s. As a result of the Algorithm 1, we obtain the transformation rules presented in Table 2.5.

We graphically illustrate this transformation process through the mapping of the temporal graph in Figure 2.7 into the property graph in Figure 2.9. The resulting property graph is composed of 4 nodes and 5 edges. We notice that for the item in Figure 2.7, 3 nodes are needed to represent its changes. Similarly, we observe that 2 edges are required to represent the changes of the *ADDTOCART* relationship.

Algorithm 1: Mapping algorithm: from conceptual temporal graph to logical property graph

```
Input: Temporal Graph: $TG = \langle E, R, T, \rho, \lambda \rangle$
Output: Property Graph $PG = \langle N, D, \eta, \Lambda, \Sigma \rangle$
/* create the Property Graph */
$N \leftarrow \emptyset$
$D \leftarrow \emptyset$
foreach temporal entity $e_i \in E$ do
    $nodeLabel \leftarrow getEntityLabel(e_i)$
    $p_{id} \leftarrow createProperty(``id", getEntityId(e_i))$
    foreach state $s_j \in S^{e_i}$ do
        $nodeProperties \leftarrow \emptyset$
        $p_{Tstart} \leftarrow createProperty(``startvalidtime", getStartValue(T^{s_j}))$
        $p_{Tend} \leftarrow createProperty(``endvalidtime", getEndValue(T^{s_j}))$
        $nodeProperties \leftarrow nodeProperties \cup \{p_{id}, p_{Tstart}, p_{Tend}\}$
        foreach attribute $a \in A^{s_j}$ do
            $p_{att} \leftarrow createProperty(``a", getAttributeValue(a))$
            $nodeProperties \leftarrow nodeProperties \cup \{p_{att}\}$
        /* create a node with a set properties */
        $N \leftarrow N \cup createNode(nodeLabel, nodeProperties)$
foreach temporal relationship $r_i \in R$ do
    $edgeLabel \leftarrow getRelationshipLabel(r_i)$
    $nodeStartLabel \leftarrow getEntityLabel(s_k)$
    $nodeEndLabel \leftarrow getEntityLabel(s_j)$
    $nodeStartProperties \leftarrow$
      $\{(``id", getEntityId(s_k)), (``startvalidtime", getStartValue(s_k)),$
    $(``endvalidtime", getEndValue(s_k))\} \cup getAllAttributeValuePair(s_k)$
    /* getAllAttributeValuePair() returns the set of all attribute-value pairs of
       a state */
    $nodeEndProperties \leftarrow$
      $\{(``id", getEntityId(s_j)), (``startvalidtime", getStartValue(s_j)),$
    $(``endvalidtime", getEndValue(s_j))\} \cup getAllAttributeValuePair(s_j)$
    $nodeStart \leftarrow matchNode(PG, nodeStartLabel, nodeStartProperties)$
    $nodeEnd \leftarrow matchNode(PG, nodeEndLabel, nodeEndProperties)$
    foreach state $s_b \in S^{r_i}$ do
        $edgeProperties \leftarrow \emptyset$
        $p_{Tstart} \leftarrow createProperty(``startvalidtime", getStartValue(T^{s_b}))$
        $p_{Tend} \leftarrow createProperty(``endvalidtime", getEndValue(T^{s_b}))$
        $edgeProperties \leftarrow edgeProperties \cup \{p_{Tstart}, p_{Tend}\}$
        foreach attribute pair $a \in A^{s_b}$ do
            $p_{att} \leftarrow createProperty(``a", getAttributeValue(a))$
            $edgeProperties \leftarrow edgeProperties \cup \{p_{att}\}$
        /* create an edge with a set properties */
        $D \leftarrow D \cup createEdge(nodeStart, nodeEnd, edgeLabel, edgeProperties)$
```

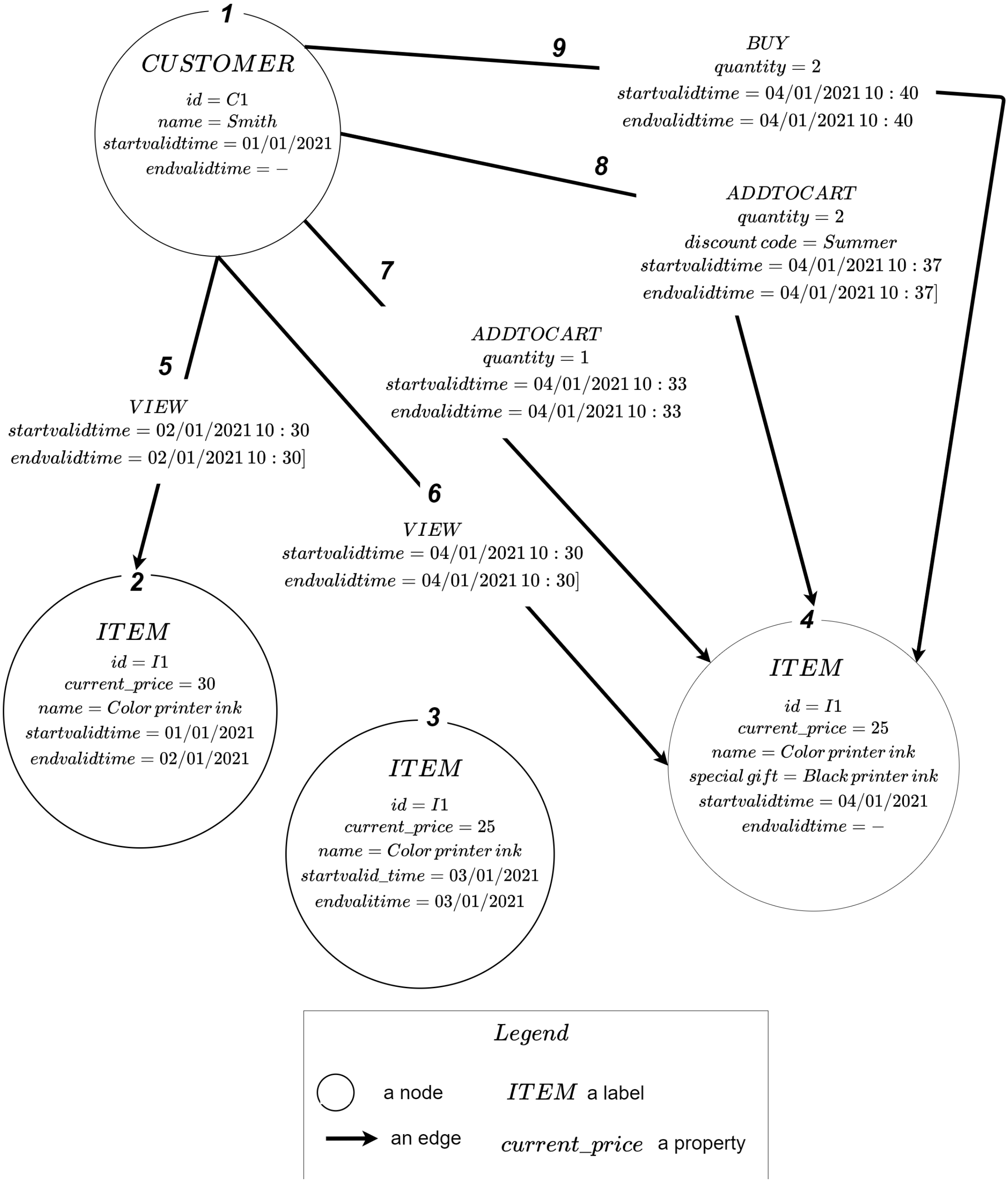

1
CUSTOMER
id = C1
name = Smith
startvalidtime = 01/01/2021
endvalidtime = −
9
BUY
quantity = 2
startvalidtime = 04/01/2021 10 : 40
endvalidtime = 04/01/2021 10 : 40
8
ADDTOCART
quantity = 2
discount code = Summer
startvalidtime = 04/01/2021 10 : 37
endvalidtime = 04/01/2021 10 : 37]
7
ADDTOCART
quantity = 1
startvalidtime = 04/01/2021 10 : 33
endvalidtime = 04/01/2021 10 : 33
5
VIEW
startvalidtime = 02/01/2021 10 : 30
endvalidtime = 02/01/2021 10 : 30]
6
VIEW
startvalidtime = 04/01/2021 10 : 30
endvalidtime = 04/01/2021 10 : 30]
2
ITEM
id = I1
current_price = 30
name = Color printer ink
startvalidtime = 01/01/2021
endvalidtime = 02/01/2021
3
ITEM
id = I1
current_price = 25
name = Color printer ink
startvalid_time = 03/01/2021
endvalitime = 03/01/2021
4
ITEM
id = I1
current_price = 25
name = Color printer ink
special gift = Black printer ink
startvalidtime = 04/01/2021
endvalidtime = −
Legend
a node
ITEM a label
an edge
current_price a property

Figure 2.9: Translation of our conceptual temporal graph in Figure 2.7 into the logical property graph

6 Implementation of a Temporal Graph Dataset: a Case Study

After discussing the conceptual representation of temporal graph data (Section 4) and the way to implement it (Section 5), we now present a case study of its implementation in a graph-oriented NoSQL data store based on the dataset of Figure 2.7 to demonstrate its technical feasibility and usability.

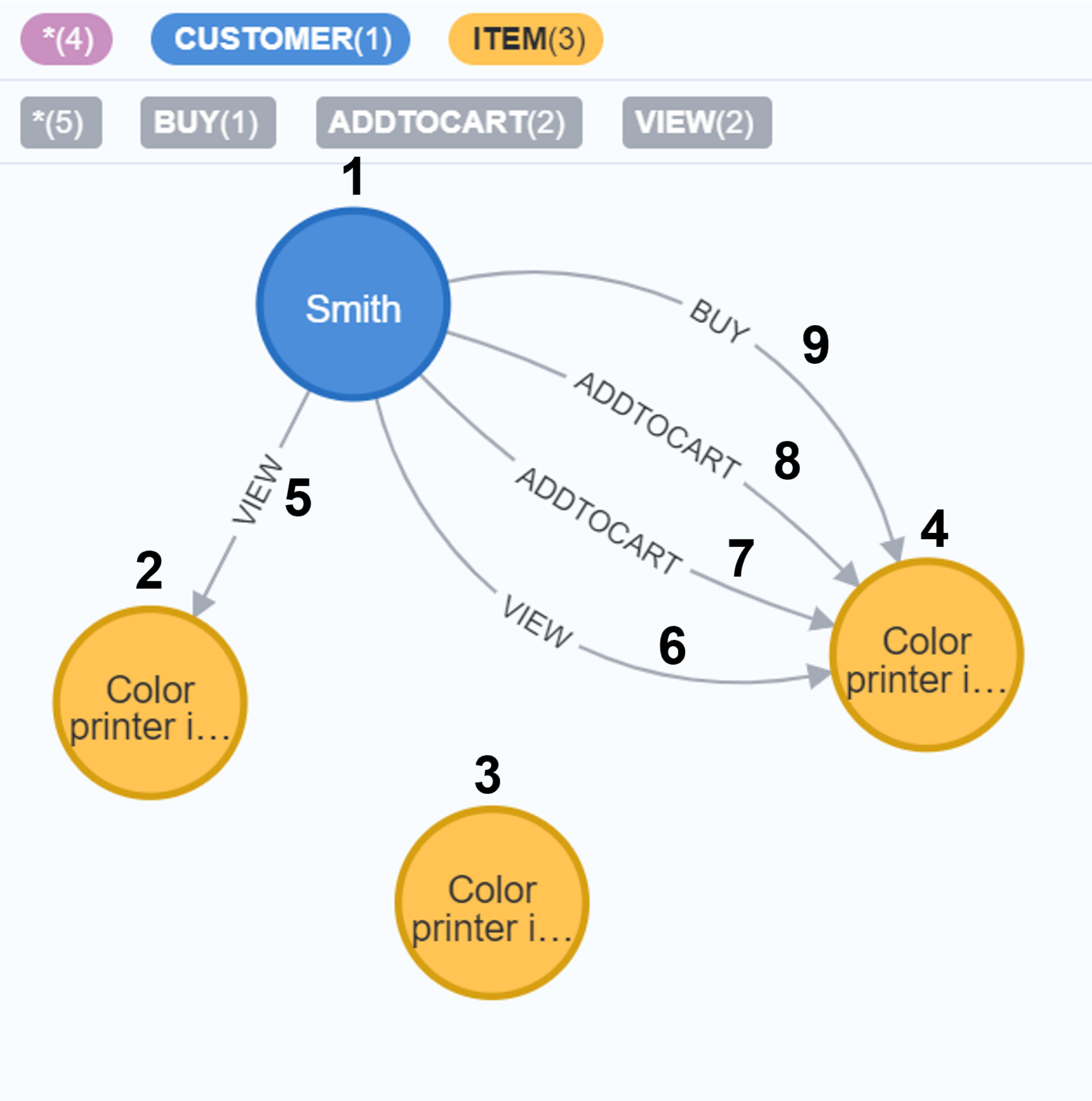
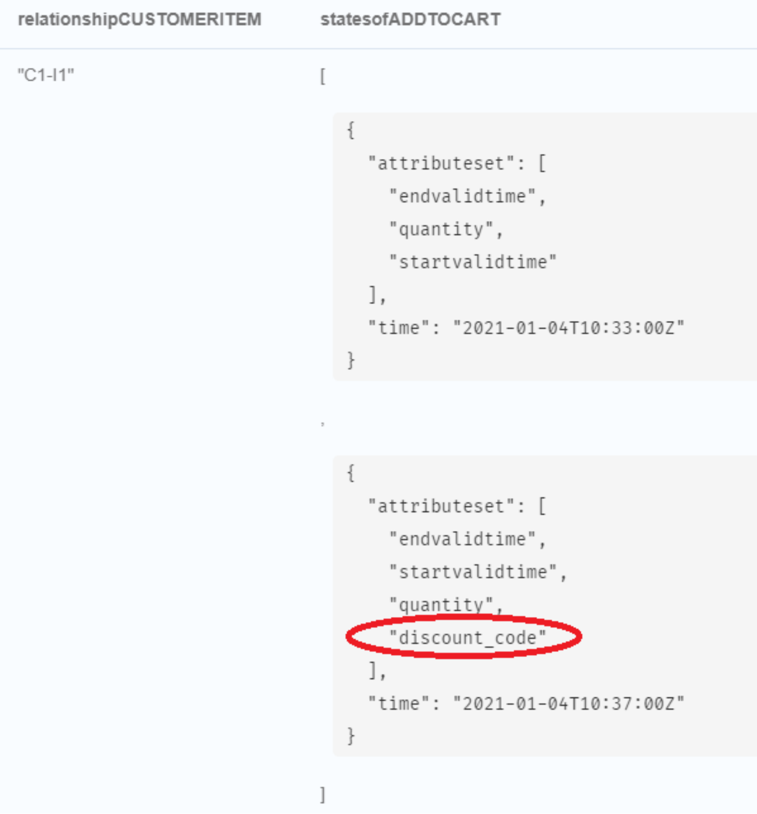


Figure 2.10: Implementation of the dataset in Figure 2.7 in Neo4j

To evaluate the technical feasibility of our conceptual model, we apply the mapping process in Algorithm 1 to implement the dataset in Fig 2.7 in Neo4j [8], a graph data store supporting the logical property graph model. Figure 2.10 presents the result of this implementation.

To evaluate the usability of our conceptual model, we identify the possible analyses on our temporal graph representation. First, the user can make a classic analysis according to the graph component only (e.g. entities or relationships). Second, the user can make an analysis according to time dimension (e.g. on continous or non continous periods etc.). Third, the user can make an analysis according to the evolution type (attribute set, attribute value or topology). The user can cross these different analysis lines to obtain more valuable insights. In the following, we propose several cross-analyses of the

[8] https://neo4j.com/



```
relationshipCUSTOMERITEM    statesofADDTOCART

"C1-I1"                     [
                              {
                                "attributeset": [
                                  "endvalidtime",
                                  "quantity",
                                  "startvalidtime"
                                ],
                                "time": "2021-01-04T10:33:00Z"
                              }
                            ,
                              {
                                "attributeset": [
                                  "endvalidtime",
                                  "startvalidtime",
                                  "quantity",
                                  "discount_code"
                                ],
                                "time": "2021-01-04T10:37:00Z"
                              }
                            ]
```

Figure 2.11: Result of business analysis B1.

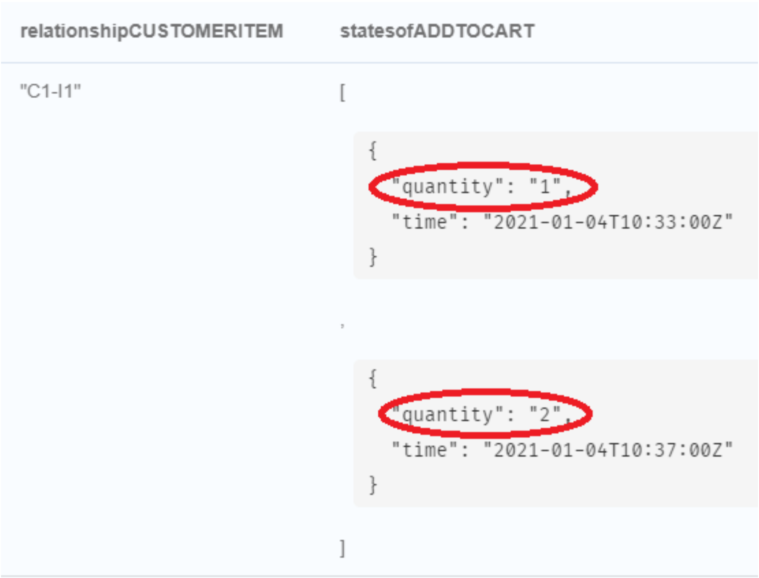
e-commerce dataset made by decision makers.

Decision makers make a first business analysis (B1). The 4th of January at 10:00, the e-commerce company announces a discount code on the website homepage for a summer promotion. To evaluate the impact of this announcement, they want to know if customers use the discount code in the hour following the announcement. This consists in analysing the addition of the attribute *discount_code* in the attribute set of the states of *ADDTOCART* relationships during the hour. This is translated in Cypher, the language query of Neo4j as follows:

```
MATCH (c:CUSTOMER)-[r:ADDTOCART]->(i:ITEM)
WHERE datetime(r.startvalidtime)< datetime("2021-01-04T12:00")
AND datetime(r.startvalidtime)>=datetime("2021-01-04T10:00")
RETURN  c.id + "-"+ i.id as relationshipCUSTOMERITEM,
collect({time:datetime(r.startvalidtime),
attributeset:keys(r)}) as statesofADDTOCART
```

As a result of B1, we obtain in Fig 2.11 the attribute set of all states of *ADDTOCART* relationships in the dataset for the period of interest. We observe that the customer identified $C1$ has used the discount code to buy the item identified $I1$ at 10:37 after the announcement of the summer promotion.

Decision makers make a second business analysis (B2). To evaluate the impact of the previous announcement, they also want to know if the quantity of items added by



```
relationshipCUSTOMERITEM     statesofADDTOCART

"C1-I1"                      [
                               {
                                 "quantity": "1",
                                 "time": "2021-01-04T10:33:00Z"
                               }
                             ,
                               {
                                 "quantity": "2",
                                 "time": "2021-01-04T10:37:00Z"
                               }
                             ]
```

Figure 2.12: Result of business analysis B2.

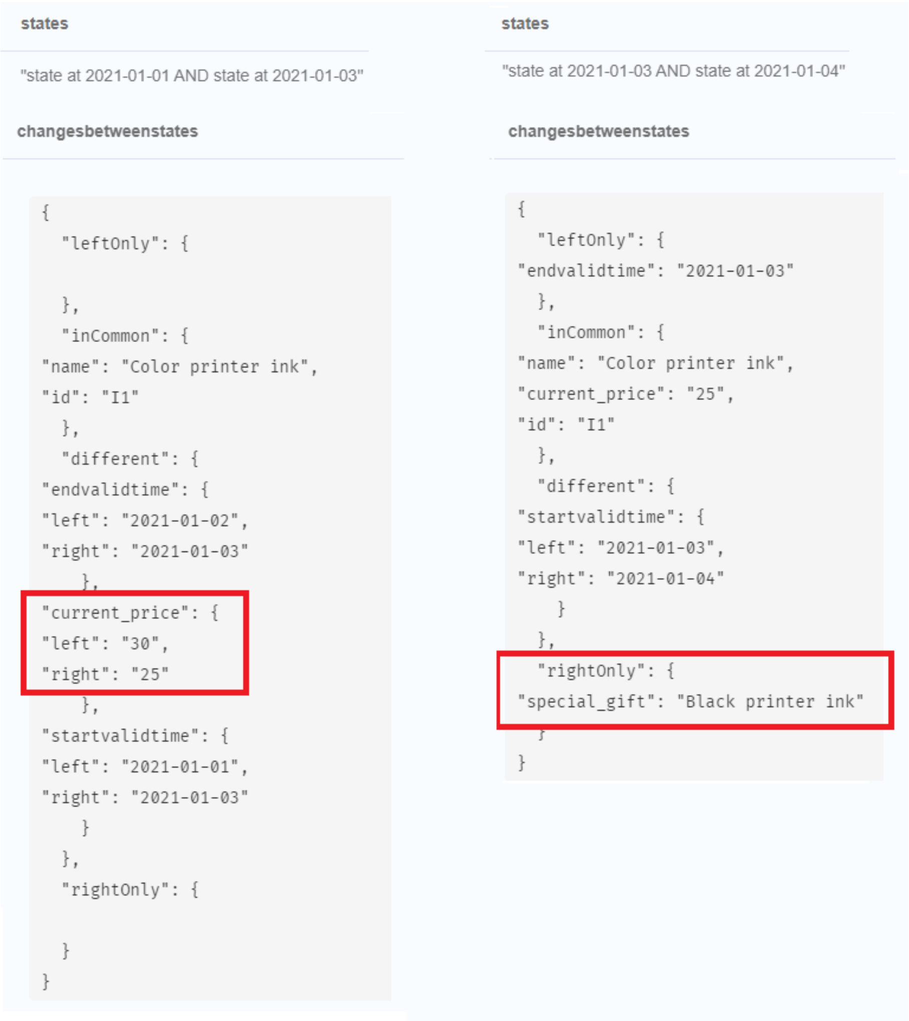
customers in their card has changed in the hour following the announcement. This consists in analysing the changes through time of the value of the attribute *quantity* of the states of *ADDTOCART* relationships during the hour. This is translated in Cypher as follows:

```
MATCH (c:CUSTOMER)-[r:ADDTOCART]->(i:ITEM)
WHERE datetime(r.startvalidtime)< datetime("2021-01-04T12:00")
AND datetime(r.startvalidtime)>=datetime("2021-01-04T10:00")
RETURN  c.id + "-"+ i.id as relationshipCUSTOMERITEM,
collect({time:datetime(r.startvalidtime), quantity:r.quantity})
as statesofADDTOCART
```

As a result of B2, we obtain in Fig 2.12 the value of the attribute *quantity* for each *ADDTOCART* relationship in the dataset for the period of interest. We observe that the customer $C1$ has updated the quantity of item $I1$ in his cart at 10:37 after the announcement of the summer promotion.

Decision makers make a third business analysis (B3). The e-commerce website records the highest number of sales on item $I1$. They want to know if this increase in $I1$'s sales is due to changes in its characteristics (new price, offer or picture etc.). This consists in analysing the changes that occurred between the states of the temporal entity $I1$. This is translated in Cypher as follows:

```
MATCH (i:ITEM)
WHERE i.id="I1"
WITH collect(i) as lists
UNWIND range(0,(size(lists)-2)) as j
RETURN "state at " + lists[j].startvalidtime+" AND "
+ "state at " +lists[j+1].startvalidtime as states,
apoc.diff.nodes(lists[j], lists[j+1]) as changesbetweenstates
```



states

"state at 2021-01-01 AND state at 2021-01-03"

changesbetweenstates

```
{
  "leftOnly": {

  },
  "inCommon": {
"name": "Color printer ink",
"id": "I1"
  },
  "different": {
"endvalidtime": {
"left": "2021-01-02",
"right": "2021-01-03"
    },
"current_price": {
"left": "30",
"right": "25"
    },
"startvalidtime": {
"left": "2021-01-01",
"right": "2021-01-03"
    }
  },
  "rightOnly": {

  }
}
```

states

"state at 2021-01-03 AND state at 2021-01-04"

changesbetweenstates

```
{
  "leftOnly": {
"endvalidtime": "2021-01-03"
  },
  "inCommon": {
"name": "Color printer ink",
"current_price": "25",
"id": "I1"
  },
  "different": {
"startvalidtime": {
"left": "2021-01-03",
"right": "2021-01-04"
    }
  },
  "rightOnly": {
"special_gift": "Black printer ink"
  }
}
```

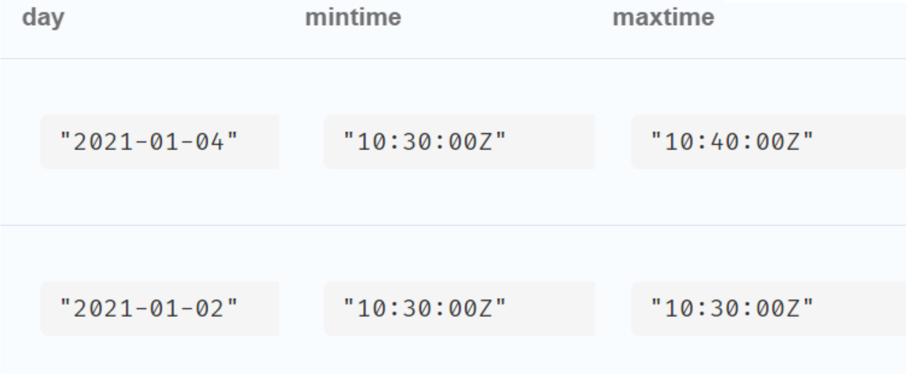
Figure 2.13: Result of business analysis B3.

As a result of B3, we obtain in Figure 2.13 the changes that occur between $I1$'s states in terms of attribute set and value. For instance, we observe that the price of $I1$ has changed from the 1st of January to the 3rd of January. Then, from the 3rd of January to the 4th of January, the attribute $special\_gift$ has been added to $I1$.

Decision makers make a fourth business analysis (B4). They want to know the time period in which customers are active on the website each day. It contributes to customer profiling to make more adapted policies in the future. This consists in analysing the addition and removal of temporal entities and relationships to see when they are connected and disconnected from the website. This is translated in Cypher as follows:

```
MATCH (c:CUSTOMER)-[r]->(i:ITEM)
RETURN date(r.startvalidtime) as day,
min(time(r.startvalidtime)) as mintime,
max(time(r.endvalidtime)) as maxtime
```

As a result of B4, we obtain, in Figure 2.14, for each day the minimum start valid time and the maximum end valid time at which customers connect to the website.



day	mintime	maxtime
"2021-01-04"	"10:30:00Z"	"10:40:00Z"
"2021-01-02"	"10:30:00Z"	"10:30:00Z"

Figure 2.14: Result of business analysis B4.

To sum up, our solution facilitates the exploration of temporal graph data through our conceptual model. Indeed, we can identify directly in the implementation of our temporal graph all data changes (Figure 2.10). Moreover, our solution provides a straightforward data restitution without introducing modelling-related concepts in the analysis results visualization (Figures 2.11, 2.12 and 2.13, 2.14). In this way, this solution makes the technical complexity transparent to non-expert users.

7 Experimental Assessments

7.1 Protocol

7.1.1 Objectives

We run two series of experiments with the two following objectives:

- to evaluate the efficiency of our proposed temporal graph model by comparing its storage and query performance to the snapshot-based graph model (Section 7.4);
- to evaluate the scalability of our proposed model by comparing its query performance on different data volumes (Section 7.5).

7.1.2 Methods

To avoid any bias in datasets, we need to include both benchmark and real datasets (Section 7.1.3). These datasets should provide different scale factors in order to measure scalability. Moreover, we need benchmark queries with a complete coverage of different analyses types (Section 7.1.4): temporal analysis (according to different time granularity), graph scope (from a single entity to the entire graph) and evolution analysis (according to topology, attribute set or attribute value).

For our two specific objectives (Section 7.1.1), we conduct two series of experiments according to the following methods. The first series consists of a comparative study of the storage (through the database size, database creation time) and query performance (through query execution times) (Section 7.4). We implement our temporal graph model and two snapshot-based models for the comparison. The second series consists of evaluating the scalability of our model during querying real-world datasets (Section 7.5).

7.1.3 Datasets

TPC-DS datasets Temporal evolutions exist in a reference benchmark available online, namely TPC-DS benchmark[9]. This benchmark is based on transaction data of a retail company. It allows us to find all the three types of evolution: (i) attribute value, (ii) attribute set and (iii) topology. We use the dataset from this benchmark to answer the objective of evaluating the efficiency of our model compared to the snapshot-based models. To do so, we transform the generated dataset from the benchmark into three datasets having our temporal graph, a classic snapshot and an optimized snapshot representations. All transformation details of the TPC-DS dataset into the three representations are available on the website `https://gitlab.com/2573869/dke_temporal_graph_experiments`. In Table 2.6, we present the results of the transformation steps: the number of vertices/edges/snapshots and the evolution types of the three TPC-DS datasets.

E-commerce dataset The E-commerce dataset has been collected from a real-world ecommerce website by RetailRocket company. It is available on Kaggle [10]. It is about customers' activity on the website (views, add to cart and transactions). The dataset includes changes over time: (i) on item characteristics with the addition of new attributes over time and the update in attribute values and, (ii) on the interactions between customers and items like clicks, add to carts and transactions. We use this dataset to answer the objective of evaluating the scalability of our model. To do so, we transform the E-commerce dataset into our temporal graph representation. All transformation details are available on the website `https://gitlab.com/2573869/dke_temporal_graph_experiments`. In Table 2.6, we present the result of the transformation steps: the number of vertices/edges and the evolution types of E-commerce dataset.

Social Experiment dataset The Social Experiment dataset has been collected from a social experiment on students from MIT who lived in dormitory Madan et al. (2012). It is available online at Reality Commons website[11]. This dataset includes changes over time: (i) on the value of the symptoms of students and (ii) on the interactions between students. We use this dataset to answer the objective of evaluating the scalability of our model. To do so, we transform the Social Experiment dataset into our temporal graph representation. All transformation details are available on the website `https://gitlab.com/2573869/dke_temporal_graph_experiments`. In Table 2.6, we present the result of the transformation steps: the number of vertices/edges and the evolution types of Social Experiment dataset.

Citibike dataset The Citibike dataset is provided by the company Citibike. The Citibike company collects data about their bicycle rentals since the year 2013 in New York City and makes them avalaible online[12]. This dataset includes bike stations and trips between these stations. We identify that the attribute set describing trips between stations has changed since May 2021. Moreover, the value of the attributes describing trips changes over time. We use this dataset to answer the objective of evaluating the scalability of our model. To do so, we transform the Citibike dataset into our

[9] `http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v2.13.0.pdf`

[10] `https://www.kaggle.com/retailrocket/ecommerce-dataset?select=item_properties_part2.csv)`

[11] `http://realitycommons.media.mit.edu/socialevolution.html`

[12] `https://www.citibikenyc.com/system-data`

Implementation	**TPC-DS: Temporal graph**	**TPC-DS: Classic snapshots**	**TPC-DS: Optimized snapshots**	**E-commerce**	**Social experiment**	**Citibike**
Objective of efficiency evaluation	YES	YES	YES	NO	NO	NO
Objective of scalability evaluation	NO	NO	NO	YES	YES	YES
# Vertices	112 897	7 405 461	5 347 477	4 821 694	33 934	2 861
# Edges	1 693 623	4 207 657	4 044 481	5 222 996	2 168 270	27 561 618
# Snapshots	N/A	60	53	N/A	N/A	N/A
Evolution types of entities	AS, AV, T	AS, AV, T	AS, AV, T	AS, AV, T	AV, T	∅
Evolution types of relationships	AS, AV, T	AS, AV, T	AS, AV, T	T	T	AV, AS, T

Table 2.6: Characteristics of datasets. *AS = Attribute Set, AV = Attribute Value, T = Topology*

temporal graph representation. All transformation details are available on the website `https://gitlab.com/2573869/dke_temporal_graph_experiments`. In Table 2.6, we present the result of the transformation steps: the number of vertices/edges and the evolution types of Citibike dataset.

7.1.4 Benchmark queries

To conduct our two series of experiments, we use the same benchmark queries to evaluate the query performance. To do so, we identify the possible query types according to all analyses axes and sub-axes that a user (e.g., a decision-maker) could have when querying temporal graph data (Khurana and Deshpande, 2016; Koloniari et al., 2013). The first analysis axis is the graph component to evaluate the cost of querying data at the level of a single entity (SE) or a set of connected entities (SU) or the entire graph (G). The second analysis axis is the evolution type to evaluate the cost of querying changes in data in terms of: attribute set (AS), attribute value (AS) or topology (T). The third analysis axis is the time scope to evaluate the cost of querying data at the level of a single time point (SP), a single interval (SI), multiple time points (MP) or multiple time intervals (MI). The fourth analysis axis is the operation type used: (i) comparison aiming at evaluating how does a graph component change over time with respect to a temporal evolution type (C) and (ii) aggregation aiming at evaluating an aggregate function (A).

We create benchmark queries in Table 2.7 by crossing the different sub-axes of analysis to distribute possible query scenarios in a balanced way. Each benchmark query represents a possible combination of analysis sub-axes. As a result, we obtain 28 queries. Finally, we translate these benchmark queries in the native query language of Neo4j: Cypher.

7.2 Technical environment

We use the same hardware configuration for the two experiments. It is as follows: PowerEdge R630, 16 CPUs x Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40Ghz, 63.91 GB. One virtual machine is installed on this hardware. This virtual machine has 6GB in terms of RAM and 100GB in terms of disk size. We install Neo4j (community version 4.1.3) as data store for our datasets on this virtual machine. To avoid any bias in the disk management and query performance, we do not use any customized optimization techniques but rely on default tuning of Neo4j. The technical details of our experiments are available on the

		Graph component	**Evolution type**	**Time scope**	**Operation type**
Q1	The descriptive attributes of an entity at X	SE	AS	SP	
Q2	The descriptive attributes of an entity at X and Y	SE	AS	MP	
Q3	The changes that occurred on the descriptive attributes of an entity between X and Y	SE	AS	MP	C
Q4	The descriptive attributes of an entity from X to Y	SE	AS	SI	
Q5	The descriptive attributes of an entity at a regular period	SE	AS	MI	
Q6	The changes that occurred on descriptive attributes of an entity from X to Y	SE	AS	SI	C
Q7	The value of an entity attribute at X	SE	AV	SP	
Q8	The value of an entity attribute at X and Y	SE	AV	MP	
Q9	The change in the value of an entity attribute between X and Y	SE	AV	MP	C
Q10	The value of an entity attribute from X to Y	SE	AV	SI	
Q11	Aggregation on the value of an entity attribute at a regular period	SE	AV	MI	A
Q12	A subgraph at X	SU	T	SP	
Q13	A subgraph at X and Y	SU	T	MP	
Q14	Aggregation on a subgraph at X	SU	T	SP	A
Q15	Aggregation on a subgraph at X and Y	SU	T	MP	A
Q16	A subgraph from X to Y	SU	T	SI	
Q17	A subgraph at a regular period	SU	T	MI	
Q18	Aggregation on a subgraph at a regular period	SU	T	MI	A
Q19	The descriptive attributes of a relationship at X	SU	AS	SP	
Q20	The descriptive attributes of a relationship at X and Y	SU	AS	MP	
Q21	The changes that occurred on the descriptive characteristics of a relationship between X and Y	SU	AS	MP	C
Q22	The descriptive attributes of a relationship from X to Y	SU	AS	SI	
Q23	The changes that occurred on the descriptive characteristics of a relationship from X to Y	SU	AS	SI	C
Q24	The value of a relationship attribute at X	SU	AV	SP	
Q25	The value of a relationship attribute at X and Y	SU	AV	MP	
Q26	The value of a relationship attribute from X to Y	SU	AV	SI	
Q27	Aggregation on the value of a relationship attribute at a regular period	SU	AV	MI	A
Q28	The state of the entire graph at X	G		SP	

Table 2.7: Benchmark queries. X and Y describe time points defined on a time unit. *SE = Single Entity, SU = Subgraph, G = Entire Graph, AS = Attribute Set, AV = Attribute Value, T = Topology, SP = Single Point, MP = Multiple Points, SI = Single Interval, MI = Multiple Intervals, C = Comparison, A = Aggregation*

website `https://gitlab.com/2573869/dke_temporal_graph_experiments`.

7.3 Summary

Regarding the first series of experiments, we create, for each of the three benchmark datasets (TPC-DS), 28 queries according to the query types we set in Section 7.1.4. We run each query ten times and take the mean time of all runs as final execution time. This makes a total of 84 queries ($28\,queries\ \times 3\,datasets$) and 840 executions ($28\,queries\ \times 3\ datasets\ \times 10\,times$).

Regarding the second series of experiments, we have three scale factors from 0.3GB to 6.7GB. We create a total of 44 queries adapted to the business contexts of the three real datasets (E-commerce, Social Experiment and Citibike). We run each query ten times and take the mean time of all runs as final execution time. This makes a total of 440 executions ($44\,queries \times 10\,times$).

7.4 Results of the efficiency evaluation of our model

For this first series of experiments, we use the three TPC-DS datasets having the following representation : our temporal graph, classic graph snapshots and optimized graph snapshots (Section 7.1.3). The classic snapshot model consists in sampling graph data at a regular time period (here we chose a month). Our optimized snapshot model consists in creating a snapshot only if it includes a change compared to a previous snapshot. We compare the storage and query performance of our temporal graph implementation to classic and optimized snapshot-based implementations through the size, creation time and query execution times in Neo4j. The query execution time is the elapsed time in seconds for processing the query. We run the 28 benchmark queries for each implementation (Section 7.3).

7.4.1 Observations of storage performance

In Table 2.8, we observe that our model reduces respectively by 12 times and 9 times the size of database instance storing classic snapshots and optimized snapshots. Moreover, the datasets based on snapshot approaches require more time to be imported since they contain more nodes and edges than our model.

7.4.2 Observations of query performance

In Figure 2.15, we observe the execution times for processing each benchmark query in Table 2.7. Queries Q1-Q6 are instantaneous (close to 0) for the three implementations. Q17-Q21 and Q27 record execution spikes for the classic and optimized snapshots implementations. The execution time of Q28 explodes for the classic snapshots and temporal graph implementations. Q28 runs out of memory for the optimized snapshots implementation. The rest of benchmark queries (Q7-Q16 and Q22-Q26) does not exceed 6 seconds for the three approaches. Overall, the execution query times of the temporal graph are lower than both snapshot-based approaches.

In Figure 2.16, we observe the average gain in execution times of the temporal graph implementation over both snapshots implementations by query types. First, we analyse the query performance of our temporal graph according to the graph component, that

Implementation	**TPC-DS : Temporal graph**	**TPC-DS: classic snapshots**	**TPC-DS: Optimized snapshots**
Size (in GB)	0,3	3,7	2,8
Creation time (in sec)	15,795	56,529	45,827

Table 2.8: Size and creation time of graph database instances in Neo4j based on benchmark datasets

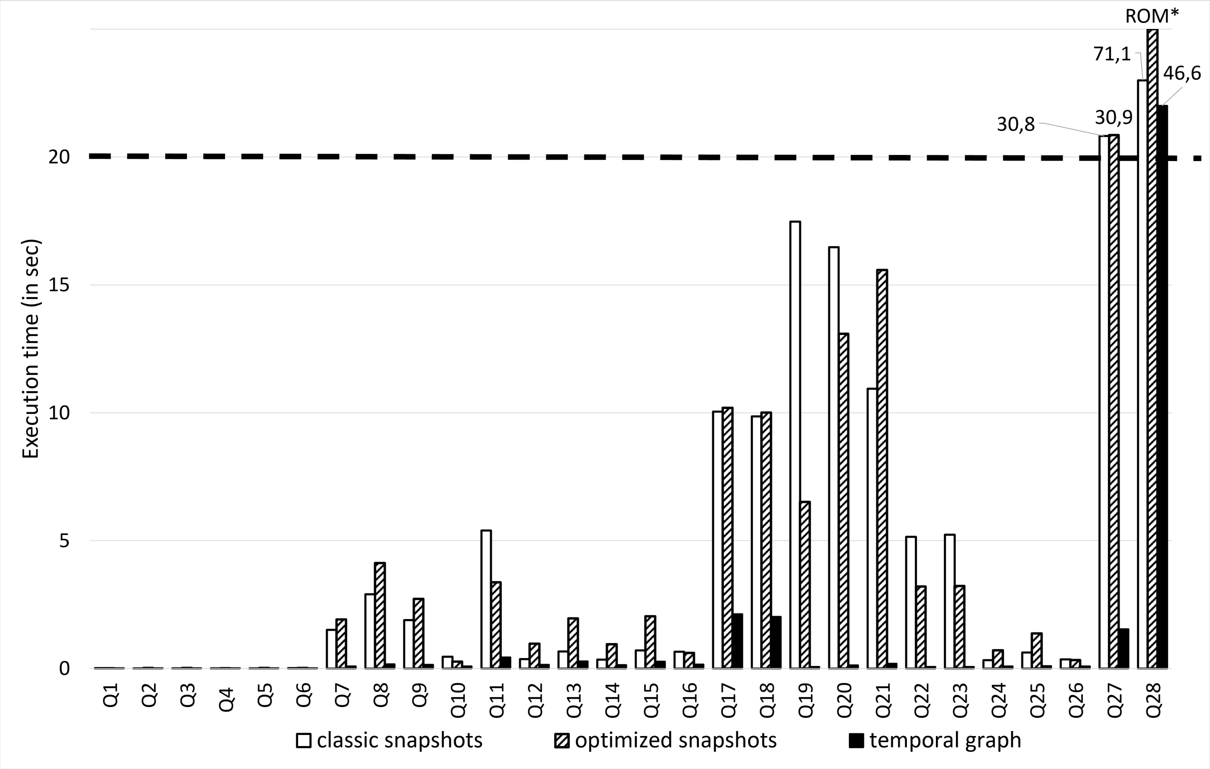


Figure 2.15: Execution times of 28 benchmark queries. *ROM = Run Out of Memory.

is requesting information at the level of a single entity (SE), a subgraph (SU) or the entire graph (G). We observe that the temporal graph implementation outperforms both snapshots approaches by saving 92%-93% of their average execution times on querying a single entity or subgraph. The gain of the temporal graph over the classic snapshots on querying the entire graph is smaller, accounting for 35%.

Second, we analyse the query performance of our temporal graph according to the evolution type, that is requesting information at the level of attribute set (AS), attribute value (AV) or topology (T). We observe that the gain of the temporal graph implementation is the highest (99%) on querying attribute set over both snapshot-based implementations. Regarding queries on attribute value, the temporal graph allows us to save 94% of average execution times over both snapshot-based implementations. The execution times gain of the temporal graph over both snapshots implementation is smaller on querying topology: 77% over classic snapshots and 81% over optimized snapshots.

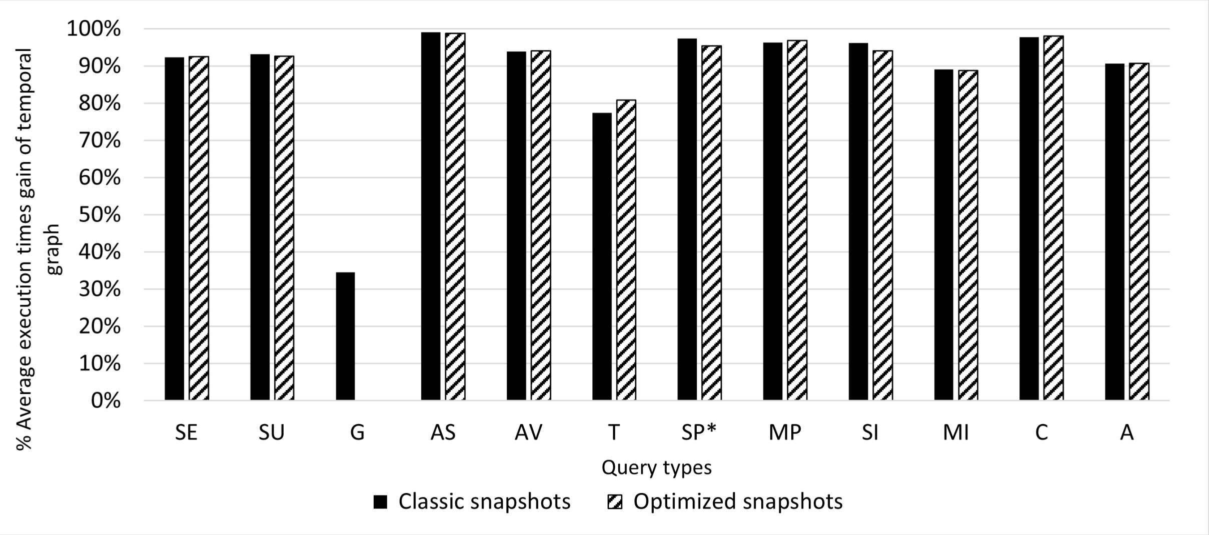


Figure 2.16: Average execution times gain (in %) of our temporal graph over classic and optimized snapshots by query types. *We do not take into account the execution time of Q28 in the computation of average execution time of SP queries because it explodes or runs out of memory for each implementation. *SE = Single Entity, SU = Subgraph, G = Entire Graph, AS = Attribute Set, AV = Attribute Value, T = Topology, SP = Single Point, MP = Multiple Points, SI = Single Interval, MI = Multiple Intervals, C = Comparison, A = Aggregation.*

Third, we analyse the query performance of our temporal graph according to the time scope, that is requesting information at the level of a single time point (SP), multiple time points (MP), a single interval (SI), or multiple time intervals (MI). We observe that there is no big difference of execution times gain of the temporal graph implementation between querying a single time point, multiple time points or single interval. The temporal graph saves from 94% to 97% of the average query execution times of snapshot-based implementations. However, the temporal graph saves less execution times on querying multiple intervals (89%).

Last but not least, we focus on the query performance of our temporal graph according to the operation type, that is comparison (C) or aggregation (A). We observe that the temporal graph saves more execution times of snapshots' implementations for processing comparison (98%) than aggregation operations (91%).

7.4.3 Discussion

The gap in the query performance between the temporal graph and the two snapshots based implementations is partly due to difference of the data volume involved in queries. The two snapshots approaches use a different time management method than our model. This leads to larger use of disk space (Table 2.8) and more time to process during querying (Figure 2.15). Across all query types, the temporal graph implementation always outperforms both snapshot-based implementations (Figure 2.16). Though the optimized snapshot implementation consumes less disk space than classic snapshot implementation, it saves almost the same average query execution times. Indeed, as time is managed differently in the two snapshot-based models, it is also queried differently. Conditions on time for classic snapshots are translated in Cypher by simple time predicates. Conversely, conditions on time in queries for the optimized snapshots are translated in Cypher by a sub-query to search for the snapshot that is the closest to a requested time. Therefore, the query performance of the optimized snapshot implementation reaches execution times almost equal or higher (e.g. Q13, Q14 or Q21) than the classic snapshot implementation.

7.4.4 Implications

The choice of a data model to manage temporal graph data impacts significantly the storage and querying efficiency. Our model has a double advantage. First, it allows to get rid of data redundancy. So it saves a significant amount of space on the disk compared to snapshots. Second, it supports efficiently a wide range of queries while keeping average execution times low. The implementation with our model allows to save up to 99% of execution times compared to both snapshot-based implementations.

7.5 Results of the scalability evaluation of our model

For this second series of experiments, we use three real datasets (Social Experiment, E-commerce and Citibike) representing three different scales of data volume and having our temporal graph representation (Section 7.1.3). We compare the query performance of the three implementation according to their scale factors: the size of database instance, the number of nodes and edges. More precisely, we analyse (i) the execution times of queries involving only entities (SE) at three different scales of the number of nodes and (ii) the execution times of queries involving relationships (SU) at three different scales of the number of edges. These two analyses allow us to get an idea of the impact of the growing size and interconnectivity of a dataset. We are not able to run the 28 benchmark queries for each implementation because each dataset does not embed all evolution types.

7.5.1 Observations

Regarding the size of each implementation, we observe in Table 2.9 that Social Experiment implementation has the smallest database instance size while Citibike has the highest one. Regarding the number of nodes and edges, we observe that E-commerce implementation is composed of the highest number of nodes (Table 2.10) while Citibike implementation is composed of the highest number of edges (Table 2.11). Regarding the average execution time of queries involving entities (SE) (Figure 2.17), the Social Experiment implementation records instantaneous execution times. On the contrary, the average execution time of queries on entities for the E-commerce implementation explodes (>30s). No queries on entities were run on the Citibike implementation. Finally, regarding the average execution time of queries involving relationships (SU) (Figure 2.18), we observe that it is globally low (at most 2s) that for the three implementations. Citibike implementation records the higher average execution time of queries involving relationships.

7.5.2 Discussion

Regarding queries on entities (SE) (Figure 2.17), the gap of execution times between the E-commerce and Social Experiment implementations is partly due to the difference in the number of nodes involved in queries. As queries on entities involve conditions on nodes, they involve a higher number of nodes during processing for the E-commerce implementation than the Social Experiment implementation. So they require more execution times to

Implementation	Social Experiment	E-commerce	Citibike
Size (in GB)	0,3	3,6	6,7

Table 2.9: Size of graph database instances in Neo4j based on real datasets.

Implementation	Social Experiment	E-commerce	Citibike
Scale factor	2	3	1
Number of nodes	33 934	4 821 694	2 861

Table 2.10: Number of nodes and scale factors of graph database instances in Neo4j based on real datasets.

process for the E-commerce implementation. The Social Experiment implementation reduces by 99% the average execution time of SE queries. Proportionally, Social Experiment implementation has 99% less nodes than the E-commerce implementation.

Regarding queries on relationships (Figure 2.18), the gap of execution times between the three implementations is partly due to the difference in the number of edges involved in queries. As queries on relationships involve conditions on edges, they involve a higher number of edges during processing for the Citibike implementation than the Social Experiment and E-commerce implementations. So they require more execution times to process for the Citibike implementation. The Social Experiment implementation has 92% less edges than the Citibike implementation. It saves 80% of the average execution time of SU queries of the Citibike implementation. The E-commerce implementation has 81% less edges than the Citibike implementation. It saves 26% of the average execution time (of SU queries) of the Citibike implementation.

7.5.3 Implications

Query execution times do not depend directly on the size of the implementation but specifically on the number of nodes and edges implemented in Neo4j. Indeed, query execution times explode with the increase in the number of nodes while stay quite low with the increase in the number of edges (i.e. the interconnectivity) in a dataset. As Neo4j is a graph-based data store, queries involving conditions on edges are more scalable compared to queries involving conditions on nodes (Vicknair et al., 2010).

8 Conclusion

To be able to explore graph data evolving over time, it is necessary to use data models providing a structure to organize them and establishing their semantics. To do so, several models are proposed in the literature. However, they come with several limitations. First, they do not associate temporal evolution at all levels of the graph. Second, they use a tracking method of graph data changes that prevents direct access to information about changes at the different graph levels. Finally, they tend to be more technically oriented by mixing implementation details in formal definitions. Regarding their implementation, they are dependent on specific technical environments. In a nutshell, current models for graph data with temporal evolution are not easily reusable to meet the needs of various users and applications.

Therefore, in this chapter, we have proposed a complete management solution for graph

Implementation	**Social Experiment**	**E-commerce**	**Citibike**
Scale factor	1	2	3
Number of edges	2 168 270	5 222 996	27 561 618

Table 2.11: Number of edges and scale factors of graph database instances in Neo4j based on real datasets

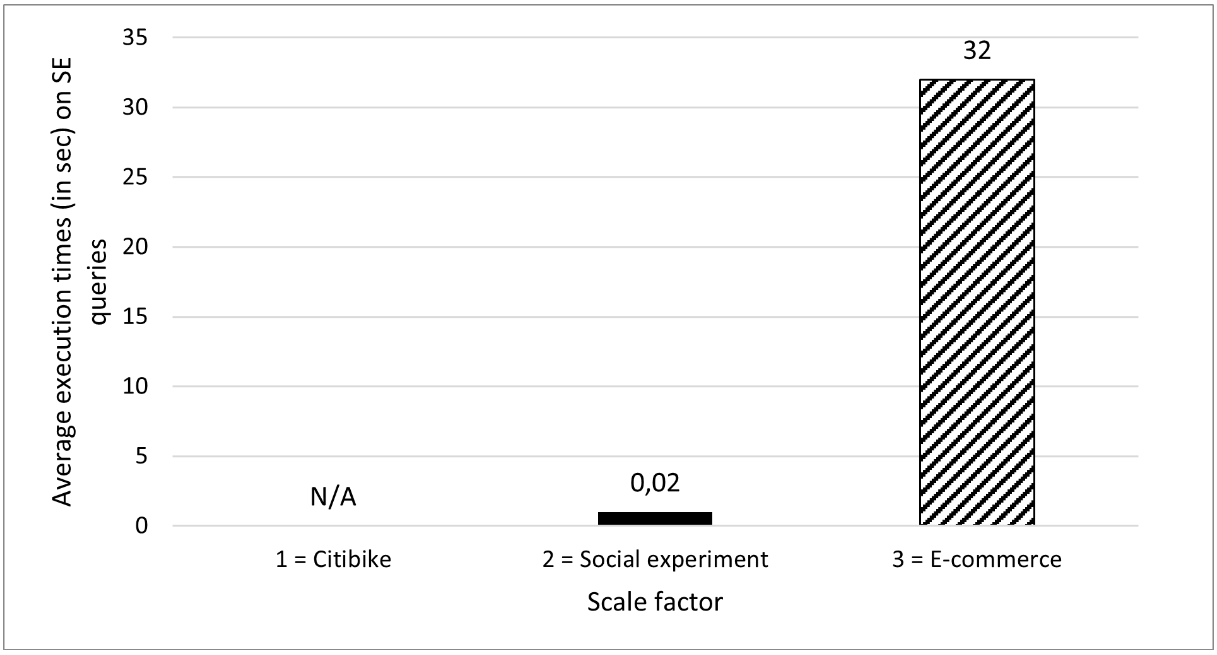


Figure 2.17: Average execution times of SE queries according to three scale factors. SE= Single Entity

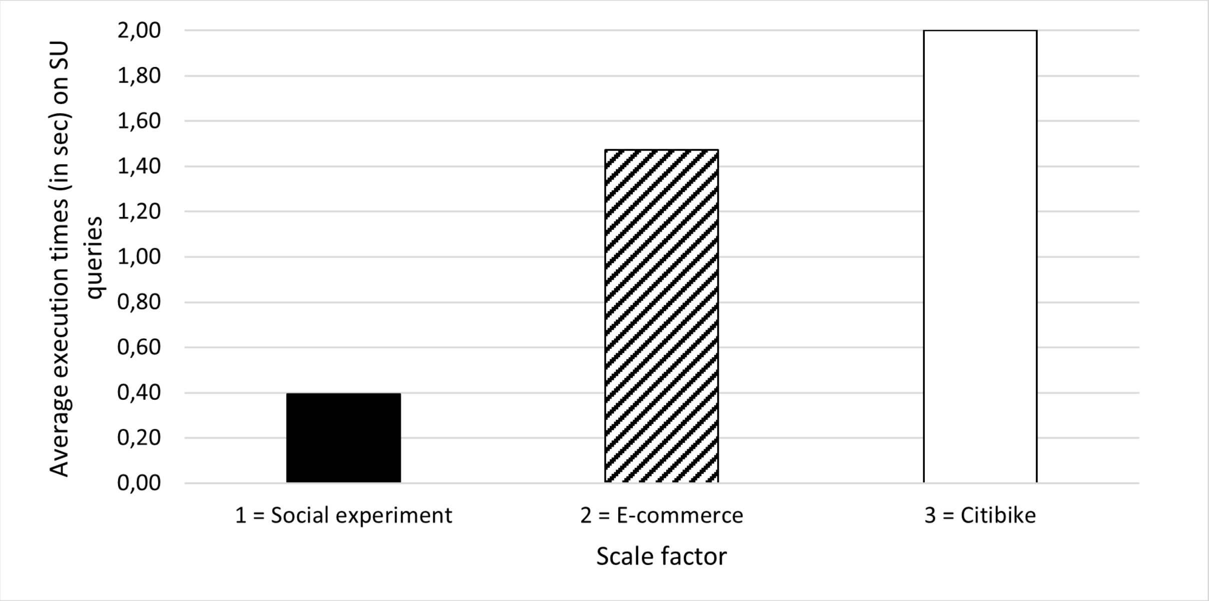


Figure 2.18: Average execution times of SU queries according to three scale factors. SU= Subgraph

data with temporal evolution: from a conceptual model, called *Temporal Graph*, to its implementation. Our conceptual model enables to represent graph data that change over time. The advantage of our model compared to existing approaches is first to be business-oriented. It provides a level of abstraction that focuses on entities and relationships and their states to reflect their evolution, without including implementation details. Second, it is generic in terms of representing all types of changes of graph data (topology, attribute set and attribute value). Finally, it provides direct access to information about graph data changes by associating a time-interval to the several levels of the graph: entities, relationships and their states.

To use our conceptual model in real business analyses, it must be transformed into a logical model before being implemented in a specific technical environment. To do so, we have proposed direct translation rules between our model and the property graph, which is commonly supported by graph-oriented NoSQL data stores. The advantage of our translation rules is that our model is directly convertible into the property graph without any specific developments. We have verified the feasibility of our model by implementing an example dataset in the Neo4j graph data store using our translation rules. Then, we verified its usability by running business analyses on evolution aspects.

We have made two experiments to evaluate the efficiency and scalability of our model. To highlight the efficiency of our model, we made a comparative study of its implementation in Neo4j with the traditional sequence of snapshots and an optimized version of snapshots based on the same dataset. We have observed that our model performs better than the sequence of snapshots by reducing 12 times disk usage and by saving up to 99% of query execution times. In comparison to the optimized sequence of snapshots, our model reduces 9 times disk usage and saves up to 99% of query execution times. In a nutshell, our model is an efficient solution for storing and querying a graph dataset with temporal evolution. To evaluate the scalability of our model, we made a comparative study of three temporal graph implementations in Neo4j based on three real-world datasets with different volumes. Globally, execution times of queries increase linearly with data volumes.

Chapter 3

Querying Temporal Graphs

Contents

1 Introduction

In the previous chapter, we have proposed the '*Temporal Graph*' which is a conceptual model providing a business-oriented representation of graph data with temporal evolution. It can be physically stored in different data stores using mapping rules we propose. The exploration of temporal graph data starts with the capability to find specific information from these data according to a user's need. This capability enables users to answer fundamental business questions such as *'What?', 'Who?', 'Where?', 'When?'.* We provide this capability through the proposition of a querying solution for graph data with temporal evolution.

Querying refers to the process of finding specific information from a data source (e.g., a graph data store) in response to users' business needs. In the literature, different approaches may be used to query the multiple dimensions of temporal graph data: query algebras (like relational algebra) (Moffitt and Stoyanovich, 2017a), textual query languages (like SQL) (Zhang et al., 2019; Debrouvier et al., 2021), or programming tools (like libraries or APIs) (Khurana and Deshpande, 2016; Massri et al., 2022). However, they lack of a generic framework which is independent of implementation environments. The absence of such a framework in current querying solutions makes it difficult for users to adapt them to different contexts.

In this chapter, the objective is therefore to propose a complete querying solution for temporal graph data. To do so, we propose a querying solution composed by three levels of abstraction: a conceptual, logical and physical level. At the conceptual level, we propose a query algebra based on the concepts of our Temporal Graph model presented in the previous chapter. It provides conceptual (i.e., business oriented) operators allowing users expressing business analyses in different contexts. The logical level translates the conceptual level into implementation-independent operators to maintain a downward compatibility with several technical environments. The physical level maps the logical level into an actual implementation within a technical environment.

The remainder of this chapter is organized as follows. In Section 2, we review existing solutions for querying temporal graph data to identify their limitations. In Section 3, we present our querying solution for temporal graph data. In Section 4, we present the experiments we carry out to evaluate the feasibility of our solution.

2 Related Work

In this section, we analyse the most recent work related to querying in temporal graph data. We classify it into three categories:

- *Query algebras,* at the logical level, consist of theoretical frameworks (or sets of operators) specifically designed to manipulate temporal graph data;
- *Extensions of textual query languages*, at the physical level, integrate new functionalities to current graph query languages to query the temporal aspects of graph data;
- *Programming tools*, at the physical level, offer pre-built functions to interact with

systems dedicated to temporal graph data.

2.1 Temporal Graph Algebras

Moffitt and Stoyanovich (2017a) propose to reuse relational algebra principles (Dignös et al., 2012). They propose a temporal property graph $G = (TV, TE)$ where TV is a set of temporal relations in which each temporal relation associates a vertex v and its attribute a during a valid time interval T and TE is a set of edges in which each edge e connects a pair of vertices v_1, v_2 associated with an attribute a during a valid time interval T. Moreover, the authors propose a temporal graph algebra (TGA) to enable users to express temporal analyses over the temporal property graph. TGA includes the *trim* operator to get vertices and edges of G which have a non-empty intersection with a user-defined time interval c, with their periods trimmed to be within c. It is denoted as follows :

$$\begin{gathered} trim_c^T(G) = (TV' = \pi_{v,T\,interesect\,c,a}(\sigma^T_{T\,overlaps\,c}(TV)), \\ (TE' = \pi_{e,v_1,v_2,T\,interesect\,c,a}(\sigma^T_{T\,overlaps\,c}(TE)) \end{gathered}$$

The *temporal subgraph matching* operator enables to get subgraphs matching a user-defined pattern, denoted as P, including a set of vertices variables and edge variables that may have temporal predicates. The operator is denoted as follows :

$$subgraph^T(P, G) = (q_v(P, G), q_e(P, G))$$

where q_v is the set of vertices that matches vertex variables and constants in P and q_e is the set of edge that matches edge variables and constants in P.

TGA's operators are composable. They take a graph as input and return a new graph. The authors develop a prototype implementation of the temporal property graph and TGA on Apache Spark.

2.2 Extensions of textual query languages

The functionalities of a textual query language are designed to align with the concepts and structures of a data model, used to represent and store data. In the context of graphs, graph query languages are commonly based on the RDF data model, such as SPARQL (W3C) [1] and RDFQL (W3C) [2], or on the property graph model, such as Cypher (Neo4j) [3], G-Core (LDBC) (Angles et al., 2018), GQL [4] and PQGL (Oracle) (Van Rest et al., 2016) (and so on). These graph query languages embed functionalities for analyses on topology and attributes. Some existing work extends the previous graph query languages for querying temporal graph data.

2.2.1 SPARQL[t]

The article of Zhang et al. (2019) presents a query language called SPARQL[t], associated with the RDFt model an extension of the RDF model. In the classic RDF model, a triple

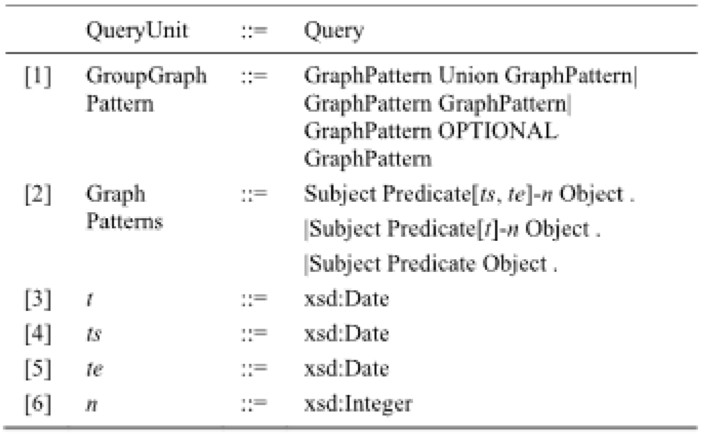
[1] https://www.w3.org/TR/rdf-sparql-query/

[2] https://www.w3.org/Submission/RDQL/

[3] https://neo4j.com/developer/cypher/querying/

[4] https://www.gqlstandards.org/what-is-a-gql-standard

is defined as (s, p, o) where s is a subject, p is a predicate, o is an object. In the RDFt model, a triple is $(s, p[t] - n, o)$ where t is a time point or a time interval ($t = [t_s, t_e]$) assigned to the predicate and n is the update count information. SPARQL[t] extends then SPARQL query language by adding several new constructors (Fig 3.1) to include RDFt expressions in the SPARQL syntax. Moreover, they propose transformation rules and algorithms to translate queries from SPARQL[t] to SPARQL and Cypher to achieve compatibility with existing query engines. Finally, they propose the prototype system in Fig 3.2 to support RDFt temporal data representation and querying.



	QueryUnit	::=	Query
[1]	GroupGraph Pattern	::=	GraphPattern Union GraphPattern| GraphPattern GraphPattern| GraphPattern OPTIONAL GraphPattern
[2]	Graph Patterns	::=	Subject Predicate[*ts*, *te*]-*n* Object . |Subject Predicate[*t*]-*n* Object . |Subject Predicate Object .
[3]	*t*	::=	xsd:Date
[4]	*ts*	::=	xsd:Date
[5]	*te*	::=	xsd:Date
[6]	*n*	::=	xsd:Integer

Figure 3.1: New syntax constructors in SPARQL[t] (Zhang et al., 2019)

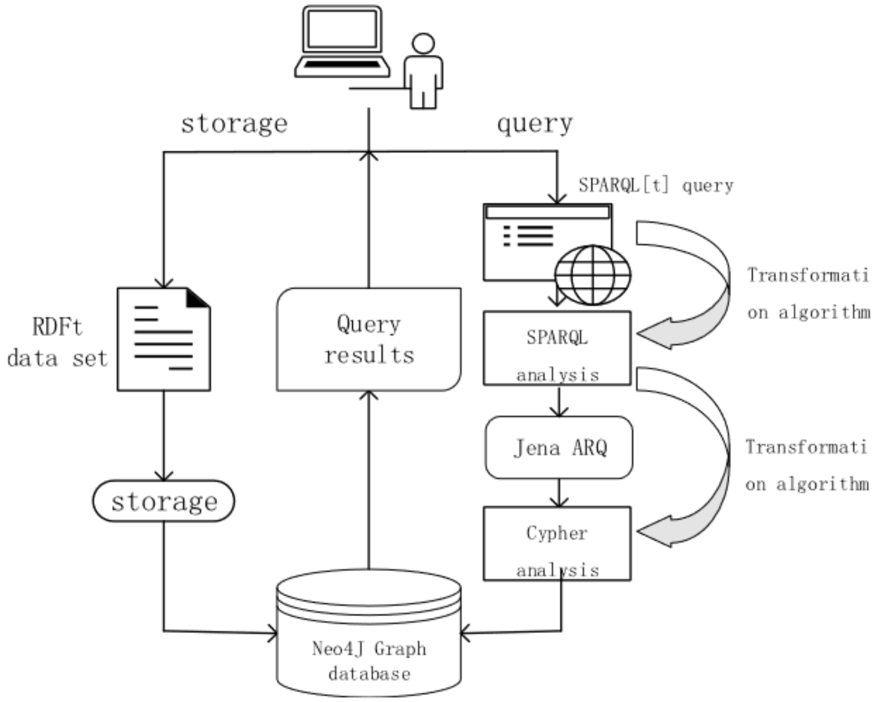
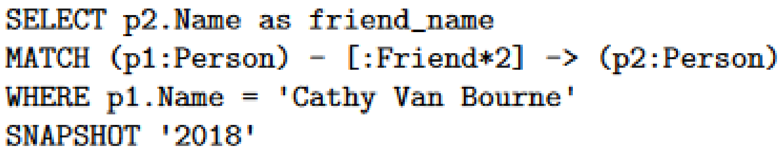


Figure 3.2: The architecture of the system proposed by (Zhang et al., 2019)

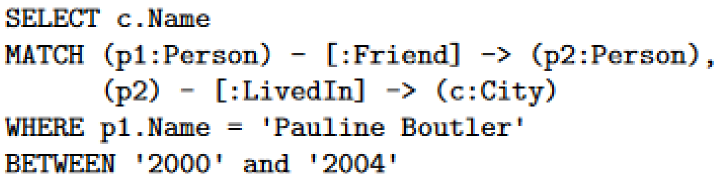
2.2.2 T-GQL

Debrouvier et al. (2021) propose the T-GQL query language associated with a temporal property graph data model, i.e., an extension of property graphs. However, contrary to the classic property graphs, this model cannot embed properties in edges, and then does not consider the temporal evolution of edges. T-GQL syntax is based on the GQL query language. To allow users to perform temporal analyses on the temporal property graph, T-GQL introduces certain temporal operators, such as the SNAPSHOT operator to return the state of the graph at a certain point in time (Fig 3.3) or the BETWEEN operator to return the state of the graph during a given interval in time (Fig 3.4). The authors implement their proposed data model and query language over a Neo4j database (Fig 3.5). The T-GQL queries are translated automatically into Cypher, Neo4j's query language using the ANTLR [5] parser generator.



```
SELECT p2.Name as friend_name
MATCH (p1:Person) - [:Friend*2] -> (p2:Person)
WHERE p1.Name = 'Cathy Van Bourne'
SNAPSHOT '2018'
```

Figure 3.3: An example of T-GQL query with the SNAPSHOT operator (Debrouvier et al., 2021)



```
SELECT c.Name
MATCH (p1:Person) - [:Friend] -> (p2:Person),
      (p2) - [:LivedIn] -> (c:City)
WHERE p1.Name = 'Pauline Boutler'
BETWEEN '2000' and '2004'
```

Figure 3.4: An example of T-GQL query with the BETWEEN operator (Debrouvier et al., 2021)

2.3 Programming Tools

Some research works propose systems dedicated to manage temporal graph data including programming tools to query temporal graph data.

2.3.1 Historical Graph Store

Khurana and Deshpande (2016) present a graph data management system, called Historical Graph Store (HGS) (Fig 3.6). It uses a temporal graph data model which is vertex-centric. The temporal graph is seen as a collection of evolving vertices over time, and edges are considered as attributes of the vertices. HGS provides the Temporal Graph Analytics Framework (TAF) which is a Java and Python based library of temporal graph operators to enable programmers to execute diverse analytical tasks. This includes the *timeslice* operator to find the state(s) of vertices or subgraphs according to a selected time point (or interval) and the *select* operator to find vertices or subgraphs according to a condition on attribute values.

[5] https://www.antlr.org/

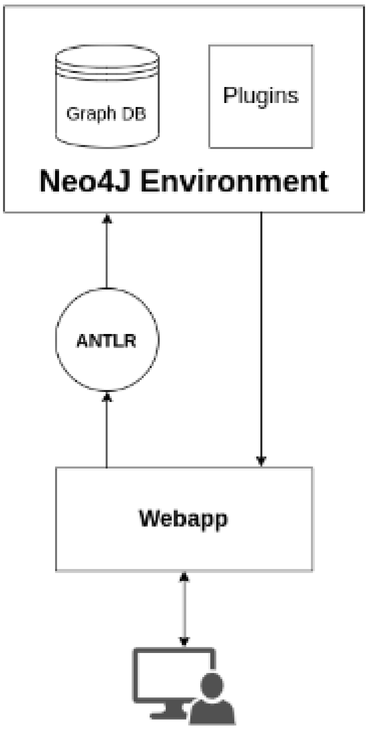


Figure 3.5: The architecture of the system proposed by (Debrouvier et al., 2021)

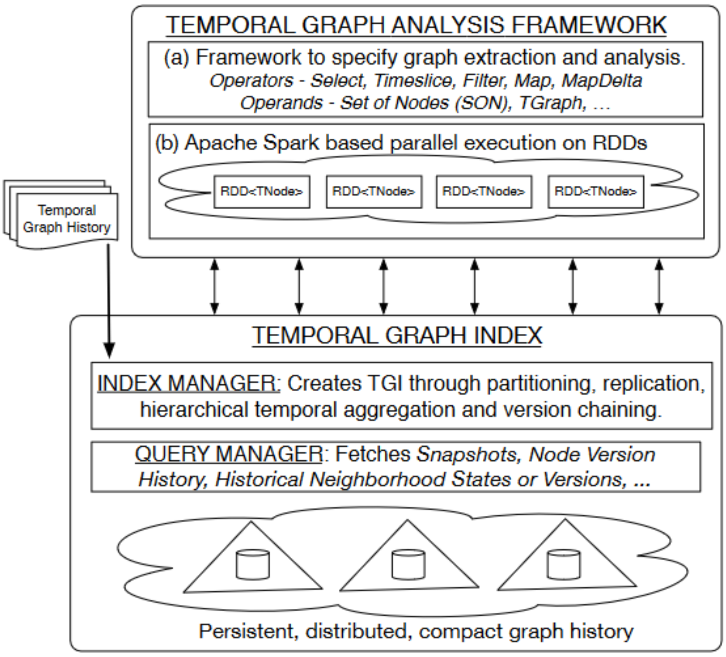

TEMPORAL GRAPH ANALYSIS FRAMEWORK
(a) Framework to specify graph extraction and analysis.
Operators - Select, Timeslice, Filter, Map, MapDelta
Operands - Set of Nodes (SON), TGraph, ...
(b) Apache Spark based parallel execution on RDDs
RDD<TNode>
RDD<TNode>
RDD<TNode>
RDD<TNode>
Temporal Graph History
TEMPORAL GRAPH INDEX
INDEX MANAGER: Creates TGI through partitioning, replication, hierarchical temporal aggregation and version chaining.
QUERY MANAGER: Fetches Snapshots, Node Version History, Historical Neighborhood States or Versions, ...
Persistent, distributed, compact graph history

Figure 3.6: The architecture of the system proposed by (Khurana and Deshpande, 2016)

2.3.2 Gradoop

Rost et al. (2021) develop the graph dataflow system, called Gradoop, for scalable, distributed analytics for large temporal property graphs (Fig 3.7). In order to express analytical problems on temporal property graphs, they provide a programming query language

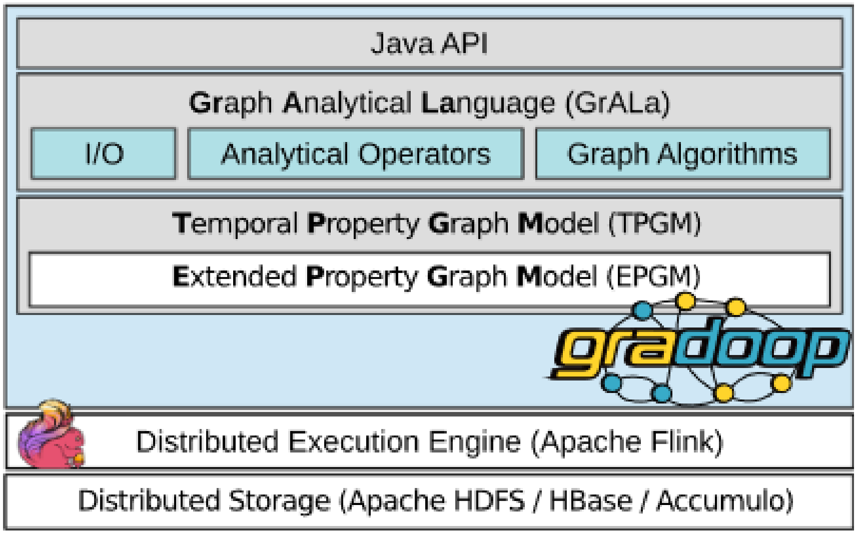
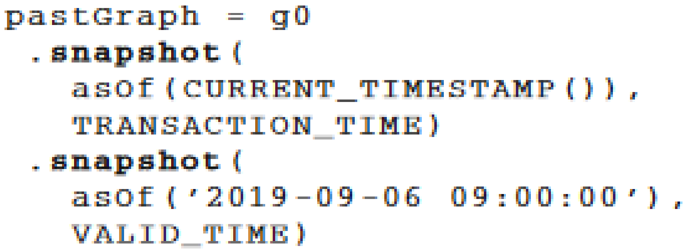

Java API
Graph Analytical Language (GrALa)
I/O
Analytical Operators
Graph Algorithms
Temporal Property Graph Model (TPGM)
Extended Property Graph Model (EPGM)
gradoop
Distributed Execution Engine (Apache Flink)
Distributed Storage (Apache HDFS / HBase / Accumulo)

Figure 3.7: The architecture of the system proposed by (Rost et al., 2021)



```
pastGraph = g0
  .snapshot(
     asOf(CURRENT_TIMESTAMP()),
     TRANSACTION_TIME)
  .snapshot(
     asOf('2019-09-06 09:00:00'),
     VALID_TIME)
```

Figure 3.8: An example of query in GrALa using the snapshot operator (Rost et al., 2021)

called GrALa composed by operators. Operators take as input and output graphs or graph collections to enable the combination of operators for advanced analysis. For instance, the *snapshot* operator extracts subgraphs (vertices and edges) that existed at a user-defined time range. It relies on temporal operators functionally similar to Allen operators (Allen, 1983) such as *asOf* (equivalent to equal) and *overlaps* (Fig 3.8). The temporal property graph model and GrALa are implemented on top of a distributed system (Apache Flink).

2.3.3 Clock-G

(Massri et al., 2022) develop a temporal graph management system called Clock-G. It is based on a data model of temporal property graph. Clock-G provides a Client API (Fig 3.2) to allow users making queries on the stored temporal graph. This API includes functionalities allowing users to make several types of temporal graph queries: point-based local queries (find a path starting from a vertex given some predicates apart from a chosen time point), range-based local queries (similar to point-based local queries but consider a time interval) and global queries (find the state of a subgraph at a given time instant or interval).

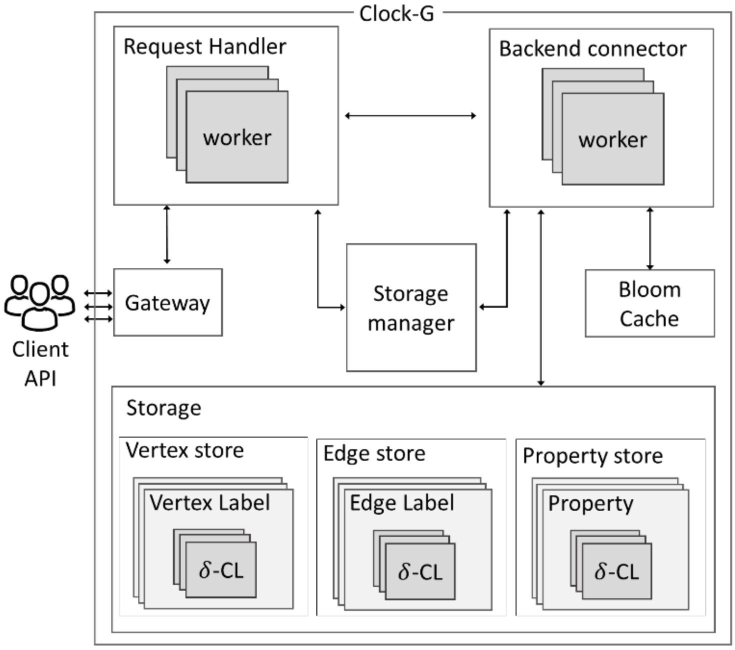


Figure 3.9: The architecture of the system proposed by (Massri et al., 2022)

2.4 Comparative Analysis of Related Work

We propose 7 comparison criteria divided into 2 categories to evaluate related work to querying temporal graph data. On the one hand, the *analysis criteria* category focuses on evaluating the querying solution's ability to formulate queries on various dimensions (Figure 3.10) and granularities of the real-world application being represented in temporal graph data. The objective is to assess if the solution is adaptable to a variety of use cases. Each criterion of this category represents therefore one dimension of temporal graph data and the different granularities of the dimension that can be explored. Each criterion in this category is assessed according to whether it is *not satisfied at all, partially satisfied, or fully satisfied.*

- **Topology**: A querying solution should support analyses on the different granularities of the topology dimension of the real-world:
  - a single entity;
  - a group of connected entities;
  - the whole set of connected entities.
- **Attribute**: A querying solution should support analyses on the attribute dimension of the real-world:
  - attributes of entities and their value;
  - attributes of relationships between entities and their value;
- **Time**: A querying solution should support analyses on the time dimension of the real-world :

- find information at different granularities of time (a time point or interval);
- find information by expressing different possible relationships between time intervals. To do so, we will refer to Allen operators (precede, meet, overlap, start, finish, equal) (Allen, 1983);
- find time-dependent information at the level of other dimensions: attribute and topology.

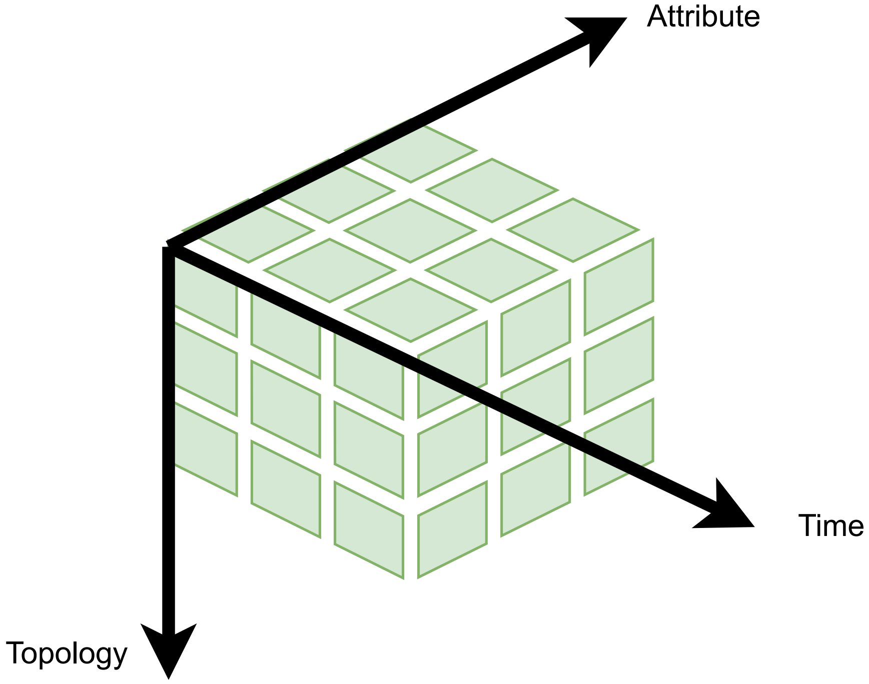


Figure 3.10: Analysis dimensions of temporal graph data

On the other hand, the *practicality criteria* category evaluates how well the querying solution is user-oriented, composable, flexible and compatible with existing technical solutions. Each criterion in this category is assessed according to the three scales: *low, medium or high.*

- **User-oriented:** A querying solution should have a clear syntax and semantics, making it straightforward for a broad range of users to express their analysis needs.
- **Composability:** A querying solution should enable to make complex analyses. To do so, it should be composable. Composability is the ability to combine and chain simple components to perform complex analyses (Angles et al., 2018). This facilitates the expression and understanding of complex analyses.
- **Flexibility:** Analysis needs may change over time. A querying solution must then be adapted to allow the addition of new functionalities to grow alongside evolving users' needs.
- **Compatibility:** A querying solution must be implementable in a technical environment without requiring custom developments. To do so, transformation methods of the querying solution are essential to be compatible with existing available query

engines to facilitate its usage.

2.4.1 Analysis criteria

We observe that data models chosen to represent temporal graph data impact the types of analyses that can be performed on them. Querying solutions of Moffitt and Stoyanovich (2017a); Rost et al. (2021); Massri et al. (2022) extend the property graph model to integrate temporal evolution of graph data. They provide functionalities to allow the different granularities of the topology and attribute dimensions to be **fully** manipulated. Querying solutions proposed by Zhang et al. (2019), Khurana and Deshpande (2016) and Debrouvier et al. (2021) are based on different data models that do not allow including multiple properties in edges. At most, they assign edges with timestamps. Therefore, they **partially** answer the attribute analysis criteria. However, all current querying solutions integrate **partially** Allen operators (at least the equality relationship) to express diverse temporal relationships over time dimension (Moffitt and Stoyanovich, 2017a; Zhang et al., 2019; Debrouvier et al., 2021; Khurana and Deshpande, 2016; Rost et al., 2021; Massri et al., 2022).

2.4.2 Practicality criteria

User-Orientation We observe that temporal graph algebras and extensions of textual query languages are solutions based on standards or well-established and widely accepted frameworks. Zhang et al. (2019) propose a textual query language based on the semantics of a standard data model concept (RDF) and on the syntax of a standard query language (SPARQL) in the Semantic Web domain. Similarly, Debrouvier et al. (2021) propose a textual query language based on GQL syntax, which is developed as a standard in the graph database community [6]. Moffitt and Stoyanovich (2017a) propose a temporal graph algebra based on the mechanisms of the relational algebra framework in relational database community. So the use of the previous solutions requires some familiarity with these standards and frameworks. Their user-orientation is therefore **medium**. Conversely, programming tools are customized temporal graph data management systems for answering specific technical issues (Khurana and Deshpande, 2016; Rost et al., 2021; Massri et al., 2022). As a result, they require a considerable effort to learn and understand the technical environment. So the user-orientation of these solutions is **low**.

Composability The temporal graph algebra in Moffitt and Stoyanovich (2017a) and the query language in the temporal graph data management system in Rost et al. (2021) are **highly** composable. The output of operators is the input of other operators, allowing for composition. It is not the case for the rest of previous works. Extensions of textual query languages have **medium** composability mainly because they have restrictions on how operators can be combined in the expression of a query (Zhang et al., 2019; Debrouvier et al., 2021). The temporal graph operators in the system of Khurana and Deshpande (2016) do not have compatible inputs and outputs. Finally, in Massri et al. (2022)'s temporal graph data management system, querying functionalities are not presented as decomposable parts.

[6] https://www.gqlstandards.org/

Flexibility The temporal graph algebra in Moffitt and Stoyanovich (2017a) is **highly** flexible. It can easily integrate new functionalities because it is completely independent of implementation details. The flexibility of extensions of textual query languages is **medium** (Zhang et al., 2019; Debrouvier et al., 2021). The evolution of their functionalities should guarantee the consistency with the query languages they are based on. Indeed, the query languages they are based on may also evolve over time. Finally, querying solutions in temporal graph data management systems are completely self-developed (Khurana and Deshpande, 2016; Rost et al., 2021; Massri et al., 2022). The integration of new functionalities depends on their technical feasibility. Therefore, they have **low** flexibility.

Compatibility Zhang et al. (2019) and Debrouvier et al. (2021) have **high** compatibility. They provide transformation mechanisms to implement directly their querying solution into diverse environments. Conversely, the rest of querying solutions previously mentioned (Moffitt and Stoyanovich, 2017a; Khurana and Deshpande, 2016; Rost et al., 2021; Massri et al., 2022) depends on a specific implementation. In other terms, they have **low** compatibility.

2.4.3 Summary

Table 3.1 provides a summary of related work of querying solutions for temporal graph data. We observe that globally existing solutions enable to make analyses on the multiple dimensions of temporal graph data. However, the level of practicality of these solutions is very variable. We notice in particular that the user audience targeted seems different according to the solution. Some solutions, such as programming tools in temporal graph data management systems, are aimed at users that have a deep technical expertise. Other solutions, such as temporal graph algebra and extensions of textual query languages, are aimed at users with some familiarity with some frameworks or standards. No querying solution is addressed to non-technical experts. The distinction in the intended user audience is primarily attributed to the fact that these solutions do not address the design of querying temporal graph data at different abstraction levels: the conceptual, logical, and physical levels. We define the latter as follows.

At the conceptual level, the focus is on how to express those business analyses without considering implementation details. This level must allow the design of composable components for two objectives: (i) to facilitate the expression of complex analyses through the combination of simple components and (ii) to anticipate the addition of new components according to the changes in users' needs. In a nutshell, the conceptual level addresses the *user-orientation, composability and flexibility* criteria of the querying solution. The compatibility aspect is not addressed at this level.

The logical level and physical levels ensure the *compatibility* of a querying solution. The logical acts as a bridge between the conceptual and physical levels. It involves translating the conceptual level into a more concrete data-oriented framework. This framework captures the structure and constraints of queries based on a data model which is technology-independent. This enables the querying solution to be implemented in different technical environments (such as database management systems) without significant modifications to the logical structure. The physical level focuses on the selection and configuration of the technical environment (such as a database management system, storage systems or

		Input	Analysis Criteria			Practicality Criteria				Abstraction Level		
		Data Model	Topology	Attribute	Time	User-Orientation	Composability	Flexbility	Compatibility	Conceptual	Logical	Physical
Temporal Graph Algebras	Moffitt and Stoyanovich (2017a)	temporal property graph	●	●	◐	MEDIUM	HIGH	HIGH	LOW		●	
Extended Textual Query Languages	Zhang et al. (2019)	RDFt	●	◐	◐	MEDIUM	MEDIUM	MEDIUM	HIGH		●	●
	Debrouvier et al. (2021)	temporal property graph	●	◐	◐	MEDIUM	MEDIUM	MEDIUM	HIGH		●	●
Programming Tools	Khurana and Deshpande (2016)	vertex-centric temporal model	●	◐	◐	LOW	LOW	LOW	LOW			●
	Rost et al. (2021)	temporal property graph	●	●	◐	LOW	HIGH	LOW	LOW			●
	Massri et al. (2022)	temporal property graph	●	●	◐	LOW	?	LOW	LOW			●

Table 3.1: Comparative Analysis of Related Work

distributed computing platforms) that will support the execution of the analysis tasks defined at the conceptual and logical levels.

We identify the temporal graph algebra presented previously (Moffitt and Stoyanovich, 2017a) at the logical level. It focuses on the formalization of analyses on temporal graph data through data-oriented concepts and in an independent implementation manner. Then, a specific implementation has been proposed for this algebra, but we do not have details on it. The extensions of textual query languages are at the logical and physical levels (Debrouvier et al., 2021; Zhang et al., 2019). They propose a logical framework to transform their proposed query languages into existing textual query languages. Moreover, they propose specific technical environments to implement their solution. We assign programming tools to the physical level (Khurana and Deshpande, 2016; Rost et al., 2021; Massri et al., 2022) since they are dependent on a system architecture. By focusing only on the logical and physical levels, the previous querying solutions do not ensure that the business needs are captured.

In conclusion, as highlighted in Table 3.1, no querying solution is sufficiently complete to satisfy all analysis and practicality criteria. Indeed, this is due to the fact that no querying solution is divided into the three abstraction levels.

3 Proposition

We propose a querying solution at three levels of abstraction - conceptual, logical and physical - to guarantee that the analysis and practicality criteria are satisfied. At the conceptual level, we propose operators that manipulate the concepts of our model of Temporal Graph. At the logical level, we propose translation rules to translate our conceptual operators into a combination of operators for querying property graphs. At the physical level, we propose an implementation methodology of the conceptual and logical level into textual query languages for property graphs.

3.1 Conceptual Level

3.1.1 Temporal Graph

We have defined a conceptual model of Temporal Graph (TG) detailed in Chapter 2. Our proposed operators manipulate the concepts of this model.

The Temporal Graph $TG = \langle E, R\rangle$ is composed of a set of entities $E = \{e_1, ..., e_g\}$ and a set of relationships $R = \{r_1, ..., r_h\}$. TG takes into account the semantics of entities and relationships through a set of labels $L^E = \{l^{E_1}, ..., l^{E_p}\}$ describing entity classes and a set of labels $L^R = \{l^{R_1}, ..., l^{R_q}\}$ describing relationship types. Each entity e_i has then an explicit semantic described by a label, $l^{E'}$ and similarly for each relationship r_i with a label $l^{R'}$. In this section, we extend the model with new functions to express the querying process.

Definition 12. *The function $\delta_{E'}$ returns, for a label $l^{E'}$, the set of entities E' having the label:*

$$\delta_{E'} : l^{E'} \to E' \textit{ where } E' \subset E$$

Definition 13. *The function $\delta_{R'}$ returns, for a label $l^{R'}$, the set of relationships R' having the label:*

$$\delta_{R'} : l^{R'} \rightarrow R' \textit{ where } R' \subset R$$

TG captures the evolution of entities and relationships over time through the concepts of *states*. Each state s of an entity or relationship is associated to a valid time interval $T^s = [t_b, t_f[$, which indicates the stability period of the state. More precisely, an entity is described by an identifier *id* and a set of states $\{s_1, ..., s_m\}$ representing its evolutions over time.

Definition 14. *The function Σ_e returns for an entity e all of its states:*

$$\Sigma_e : e \rightarrow \{s_1, ..., s_m\}$$

An entity may have different types of relationship with other entities. A couple of entity states (s_k, s_j) may therefore be linked by relationship r with a type $l^{R'}$.

Definition 15. *The function ρ returns, for two entity states s_k and s_j and a relationship label $l^{R'}$ connecting these two, a temporal relationship r if it exists:*

$$\rho : s_k \times s_j \times l^{R'} \rightarrow r$$

Similarly, as entities, a relationship r is composed by a set of states $\{s_1, ..., s_n\}$ to represent its evolutions over time.

Definition 16. *The function Σ_r returns for a relationship r all of its states:*

$$\Sigma_r : r \rightarrow \{s_1, ..., s_n\}$$

3.1.2 Running Example

We propose an example of a TG in Figure 3.11 to analyse the interactions between individuals of a university in the context of a disease spreading over a time period of 6 days. In the university, we distinguish students from teachers. Each entity (i.e., a student or a teacher) is represented in the TG by a grey vertex labelled with its semantics (*student* or *teacher*). Moreover, we distinguish two types of interactions between students and teachers: virtual contact (*call* relationship) and physical contact (*socialize* and *attend class* relationships). For instance, the set of students is $\delta_{E'}(STUDENT) = \{15, 76\}$. In other terms, the TG is composed by students identified 15 and 76. Taking into account the semantics during analyses can make a great difference. In this context, we can ignore *virtual contacts* and put more emphasis on *physical contacts* of teachers and students, who generally meet more people during classes. For instance, the set of physical contacts 'socialize' between entities is $\delta_{R'}(SOCIALIZE) = \{(s_1, s_3), (s_2, s_3)\}$. In other terms, this is composed by relationships identified by the couple of entity states (s_1, s_3) and (s_2, s_3).

The health status of students and teachers can change over time due to the disease transmission. Each entity (grey vertex) is then composed of one or several states (white vertices). For instance, the student 76 has two different health status with $\Sigma_e(id = 76) = \{s_1, s_2\}$. In the state 1, he has no fever and no cough from day 1 to day 2, and in the state 2, he has fever and cough from day 3 to day 6.

Moreover, new interactions between students and teachers may appear and disappear over time. Each interaction represents a state of a labelled relationship between two entity states. An interaction is illustrated by a white rectangle. For instance, if we look

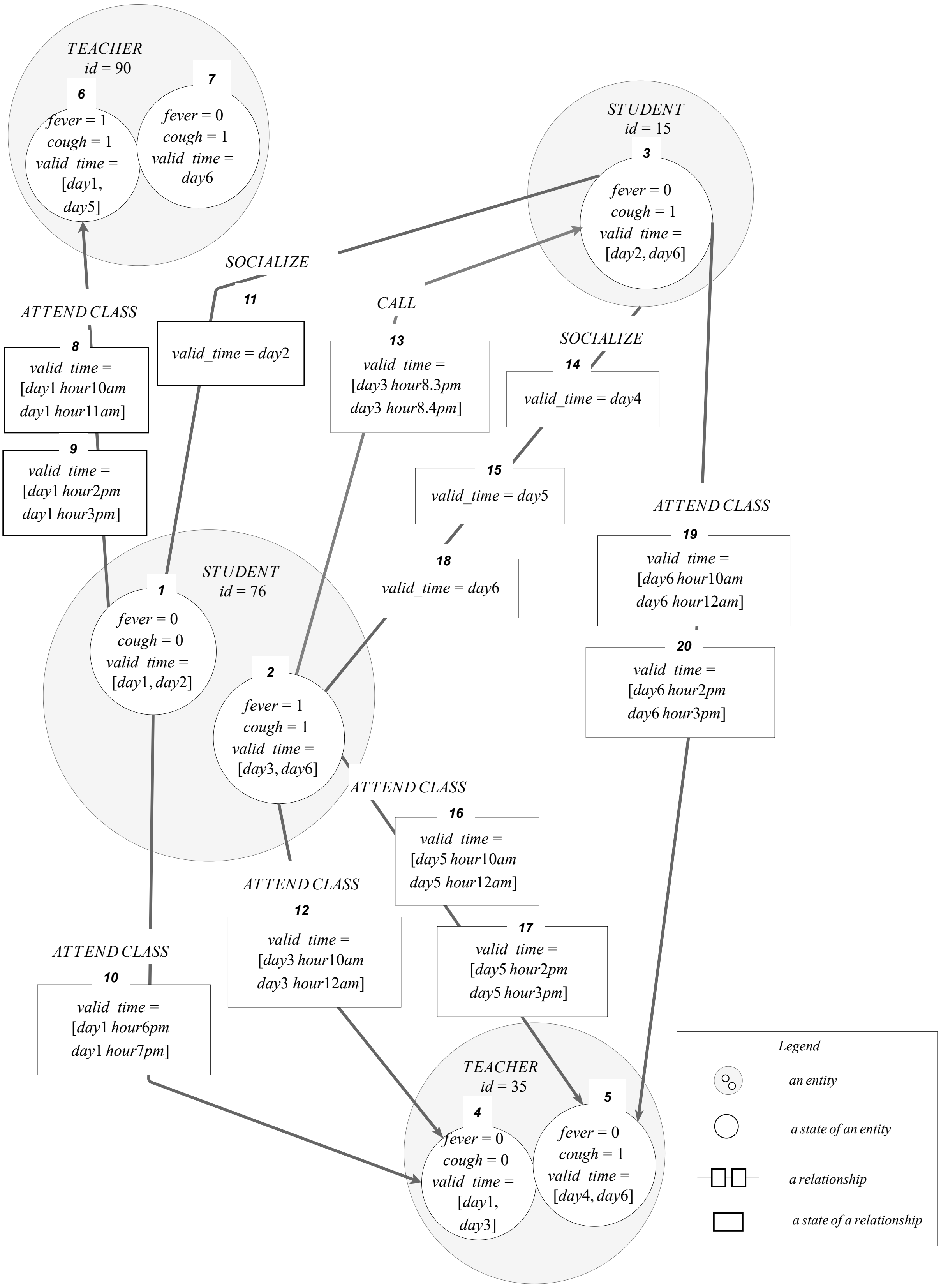


Figure 3.11: An example of temporal graph

at the contacts between the student 15 in the state 3 and the student 76 in the state 2, they interact three days during the studied period (rectangles numbered 14, 15 and 18).

Indeed, we have $\Sigma_r(s_3, s_2) = \{s_{14}, s_{15}, s_{18}\}$.

3.1.3 Operators for Querying Temporal Graph

The objective of our querying solution is to enable users to perform fundamental analyses over the topology, attribute, and time dimensions of a real-world application that involves interconnected entities. To achieve this goal, we define in this section two conceptual operators: one dedicated to manipulate the attribute and time dimensions and another to manipulate the topology dimension. Indeed, the attribute and time dimensions are manipulated in the same manner, unlike the topology dimension. These operators are based on our conceptual TG model presented in Section 3.1.1.

In the context of dynamic applications, users may want to find information dependent on time and attributes of entities (or relationships). Generally, for classic graphs, there exist functionalities in existing querying solutions to filter data according to conditions on vertex (or edge) attributes. In other terms, they allow extracting data limited only to those vertices (or edges) that satisfy the condition. Below, we propose the operator $match_{predicates}$ capable of filtering the TG according to user-defined conditions on the attributes and valid time intervals of entities (or relationships).

Definition 17. *The $match_{predicates}$ operator is used to extract a subgraph TG_{output} from an input temporal graph TG_{input} that satisfies a combination π of user-defined predicates. It is denoted as follows:*

$$match_{predicates} : TG_{input} \times \pi \rightarrow TG_{output}$$

where $\pi = \{\pi_1 \,\beta\, \pi_2 \,\beta ... \beta\, \pi_{n-1} \,\beta\, \pi_n\}$ where each $\pi_i \in \pi$ is a predicate and β is a connector. A predicate $\pi_i = X.c$ expresses a condition (or criteria) denoted as c that is assigned to a given subject X in TG_{input}. There are two types of predicates:

- *an **attribute predicate** denoted as, $X.a\,\theta\,w$. It expresses a condition on the attribute a of a subject X, which is a comparison of the value of an attribute a and a user-defined value w using the comparison operator θ. Different comparison operators are presented in Table 3.2 to express different meanings of the relationship between the two attribute values.*

- *a **temporal predicate** denoted as, $X.T\,\alpha\,T_u$. It expresses a condition on the valid time interval T of a subject X, which is a comparison of the value of the valid time interval T and a user-defined valid time interval T_u using the Allen operator θ (Allen, 1983). We integrate all Allen operators to express different temporal relationships (Table 3.2).*

A subject X is a set of entities or relationships represented by $X \subseteq (E \cup R)$. A subject X can be one of the following options:

- *all entities of the input TG denoted as E;*

- *all relationships of the input TG denoted as R;*

- *all entities belonging to the same entity class of the input TG denoted as $l^{E'}$;*

- *all relationships belonging to the same relationship type of the input TG denoted as*

Table 3.2: Predicate Types. In Allen operators, $T = [t_s, t_f[$ is a valid time interval. $T_u = [x, y[$ is a user-defined time interval where variables x and y are time instants.

Predicate Type	Predicate	Description
Attribute	$a = w$	an attribute a has a value equal to w
	$a <> w$	an attribute a has a value different from w
	$a < w$	an attribute a has a value less than w
	$a \leq w$	an attribute a has a value less or equal than w
	$a > w$	an attribute a has a value greater than w
	$a \geq w$	an attribute a has a value greater or equal than w
	$a\, BETWEEN\, w_1\, AND\, w_2$	an attribute a has values within a given range $[w_1, w_2]$
	$a\, IS\, NULL$	an attribute a is null
	$a\, IS\, NOT\, NULL$	an attribute a is not null
	$a\, STARTS\, WITH\, w$	an attribute a has a string value that starts with w
	$a\, END\, WITH\, w$	an attribute a has a string value that ends with w
	$a\, CONTAINS\, w$	an attribute a has a string value that contains w
Temporal	$T < T_u$	T precedes T_u $t_f < x$
	$T_u > T$	T_u is preceded by T $x > t_f$
	$T\, m\, T_u$	T meets T_u $t_f = x$
	$T \circ T_u$	T overlaps T_u $(t_s < y)$ AND $(t_f \geq x)$
	$T\, s\, T_u$	T starts T_u $t_s = x$
	$T\, d\, T_u$	T during T_u $(t_s \geq x)$ AND $(t_f < y)$
	$T\, f\, T_u$	T finishes T_u $t_f = y$
	$T\ =\ T_u$	T equals T_u $(t_s = x)$ AND $(t_f = y)$

$l^{R'}$;

A connector β allows combining different predicates p to evaluate if their combination is true or false. Here are the list of connectors to combine predicates:

- *the AND operator combines multiple predicates, such as $\pi = \{\pi_1\, AND\, \pi_2\, AND ... AND\, \pi_{n-1}\, AND\, \pi_n\}$. It allows evaluating if all predicates of the combination are true;*
- *the OR operator combines multiple predicates, such as $\pi = \{\pi_1\, OR\, \pi_2\, OR ... OR\, \pi_{n-1}\, OR\, \pi_n\}$. It allows evaluating if at least one of the predicates of the combination is true;*

Table 3.3: Matching predicates operator

Operator:	$match_{predicates}(TG_{input}, \pi)$
Input:	an input graph TG_{input} a combination of predicates π
Output:	a subgraph TG_{output}
Actions:	1. $TG_{output} \leftarrow \langle \emptyset, \emptyset \rangle$ 2. If $AND \in \pi$ 3. If $\forall \pi_i \in \pi$, $evaluate_{predicate}(TG_{input}, \pi_i) \neq \emptyset$ then 4. For each $\pi_i \in \pi$ 5. $TG_{output} \leftarrow TG_{output} \cap evaluate_{predicate}(TG_{input}, \pi_i)$ 6. End For 7. End If 8. Else If $OR \in \pi$ 9. If $\exists \pi_i \in \pi$ that $evaluate_{predicate}(TG_{input}, \pi_i) \neq \emptyset$ 10. For each $\pi_i \in \pi$ that $evaluate_{predicate}(TG_{input}, \pi_i) \neq \emptyset$ 11. $TG_{output} \leftarrow TG_{output} \cup evaluate_{predicate}(TG_{input}, \pi_i)$ 12. End For 13. End If 14. Else If $NOT \in \pi_i$ 15. $TG_{output} \leftarrow TG_{input} \setminus match_{predicates}(TG_{input}, \pi)$ 16. End If

- *the NOT operator negates an individual predicate or a combination of predicates, such as $NOT(\pi_1\, \beta ... \beta\, \pi_n\}$. It allows evaluating if the predicate or the combination of predicates is false.*

Remark 2. *It is possible to select a specific entity or group of entities by using an attribute predicate in which their identifier id^{e_i} is the attribute. Let us consider the TG in the running example (Section 3.1.2). We can analyse a specific student identified as 76 by using the predicate $(STUDENT.id = 76)$ in the $match_{predicates}$ operator.*

The algorithm of the execution of the $match_{predicates}$ operator is presented in Table 3.3. According to the connector used in the combination of predicates, the actions done on the TG_{input} to extract the output subgraph TG_{output} are different. It is important to notice that the algorithm uses the $evaluate_{predicate}$ operator which enables, for a given predicate, to extract the elements of the subject (here, states of entities or relationships) that satisfy the condition (Table 3.4).

Remark 3. *A combination of predicates π may include different connectors. In that case, users have to parse the combination of predicates π to identify parenthetical expressions and their respective sub-predicates. They have to identify open and close parentheses to determine the scope of each sub-predicate. Then, the $match_{predicates}$ operator will evaluate the sub-predicates within each parenthetical expression separately, respecting the order of operations dictated by parentheses. Let us consider $\pi = \pi_1 AND \pi_2 OR \pi_3 AND \pi_4 OR \pi_5 OR \pi_6$. Before applying the $match_{predicates}$ operator, by applying a parsing of π, we can have $\pi = \pi_1 AND (\pi_2 OR \pi_3) AND (\pi_4 OR \pi_5 OR \pi_6)$.*

Example 1. *Let us consider the TG of the running example (Section 3.1.2). Users want*

Table 3.4: Evaluate predicate operator

Operator:	$evaluate_{predicate}(TG_{input}, \pi_i)$
Input:	an input graph $TG_{input} = \langle E, R \rangle$ a predicate $\pi_i = X.c$
Output:	a subgraph $TG_{output} = \langle E', R' \rangle$
Actions:	1. $E' \leftarrow \emptyset$ 2. $R' \leftarrow \emptyset$ 3. If $X \subseteq E$ then 4. For each $e \in X$ 5. Get the states of e 6. For each $s_i \in \Sigma_e(e)$ 7. If $s_i.c$ is false then 8. $E' \leftarrow E \setminus \{s_i\}$ 9. End If 10. End For 11. End For 12. Else If $X \subseteq R$ then 13. For each $r \in X$ 14. Get the states of r 15. For each $s_i \in \Sigma_r(r)$ 16. If $s_i.c$ is false then 17. $R' \leftarrow R \setminus \{s_i\}$ 18. End If 19. End For 20. End For 21. End If 22. $TG_{output} \leftarrow \langle E', R' \rangle$ 23. Return TG_{output}

to know who are the persons who had fever during a period overlapping the period from day 3 to day 5. This analysis includes a condition on the health state of all entities and a condition on the occurrence of the state. We can then apply a $match_{predicates}$ *operation as follows:*

$$TG_{output} = match_{predicates}(TG, \{E.valid\_time \circ [day3, day5]\ AND\ E.fever = 1\})$$

For each predicate $E.valid\_time \circ [day3, day5]$ *and* $E.fever = 1$*, the* $match_{predicates}$ *operator first extracts the subgraph of* TG *that satisfies the predicate using the* $evaluate_{predicate}$ *(Table 3.4). Second, since the two predicates are combined with the* AND *connector, the* $match_{predicates}$ *operator makes the intersection between the two subgraphs obtained in the previous operation (Table 3.3). As shown in the Fig 3.12, the result is the non-connected subgraph composed of the states of entities satisfying both predicates.*

Example 2. *Let us consider the* TG *of the running example (Section 3.1.2). Users want to find all socializations between students at day 2 or at day 3. This analysis includes a condition on the occurrences of relationships labelled* $SOCIALIZE$*. So we use a* $match_{predicates}$ *operation as follows:*

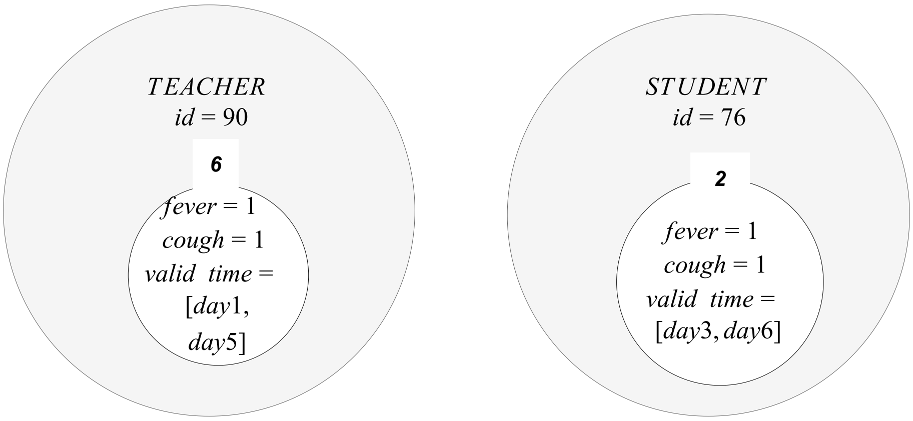


Figure 3.12: Result of the Example 1

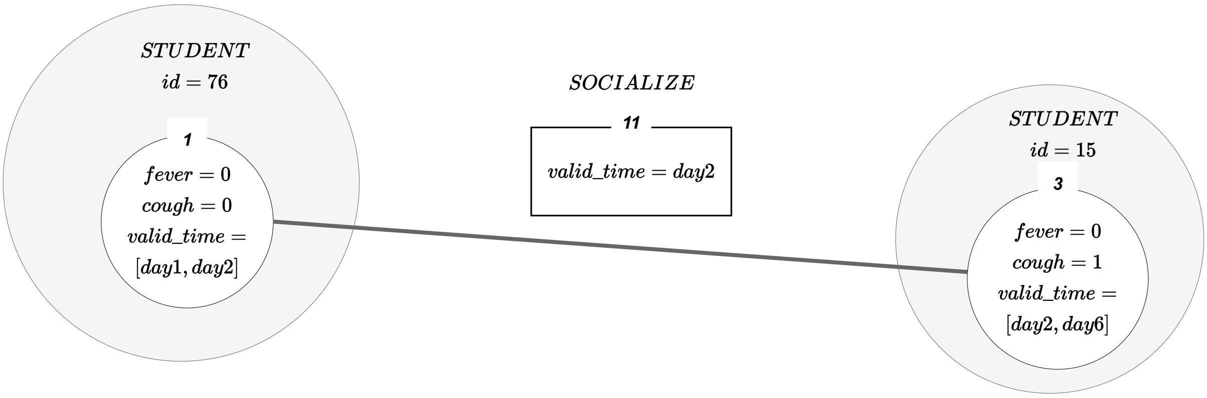


Figure 3.13: Result of the Example 2

$$TG_{output} = match_{predicates}(TG, \{SOCIALIZE.valid\_time = day2\, OR\, SOCIALIZE.valid\_time = day3\})$$

For each predicate $SOCIALIZE.valid\_time = day2$ and $SOCIALIZE.valid\_time = day3$, the $match_{predicates}$ operator first extracts the subgraph of TG that satisfies the predicate using the $evaluate_{predicate}$ (Table 3.4). Second, since the two predicates are combined with the OR connector, the $match_{predicates}$ operator makes the union between two subgraphs obtained in the first operation (Table 3.3). As shown in the Fig 3.13, the result is the subgraph TG_{output} composed of the states of relationships that satisfy at least one of the predicates. No relationship state in TG satisfies the second predicate.

Remark 4. *In the case an analysis involves a specific graph structure, such as teachers connected to students, the $match_{predicates}$ operator applies operations on each element of the graph structure. It cannot manipulate directly the graph structure to extract the connections between the elements. For instance, consider that users want to find the following information: the teachers who had fever from day 3 to day 5 and their interactions with students during day 1. If we use the $match_{predicates}$ operator for this analysis, it will be written as follows:*

$$TG_{output} = match_{predicates}(TG, \{TEACHER.valid\_time\, d\, [day3, day5]\; AND\, ATTEND\_CLASS.valid\_time\, d\, day1\})$$

Considering the interconnectivity of entities, users may want to find specific patterns,

Table 3.5: Pattern matching operator

Operator:	$match_{pattern}(TG_{input}, P)$
Input:	a pattern $P = (l^{E_i}, l^{R_j}, l^{E_k}, ..., l^{R_m}, l^{E_n})$
Output:	a subgraph TG_{output}
Actions:	1. Initialize the position of a label in P to $i = 1$ 2. While $i < size(P) - 1$ 3. $TG_{output} \leftarrow \langle \emptyset, \emptyset \rangle$ 4. Extract entities labelled l_i from TG_{input} using $\delta_{E'}(l_i)$ 5. Extract entities labelled l_{i+2} from TG_{input} using $\delta_{E'}(l_{i+2})$ 6. For each $e_h \in \delta_{E'}(l_i)$ 7. For each $e_f \in \delta_{E'}(l_{i+2})$ 8. Get the states of e_h using $\Sigma_e(e_h)$ 9. Get the states of e_f using $\Sigma_e(e_f)$ 10. For each $s_j \in \Sigma_e(e_h)$ 11. While there exists a state $s_k \in \Sigma_e(e_f)$ such that $\rho(s_j, s_k, l_{i+1}) \neq \emptyset$ 12. $TG_{output} \leftarrow TG_{output} \cup \langle \{s_j, s_k\}, \{\rho(s_j, s_k, l_{i+1})\} \rangle$ 13. End while 14. End For 15. End For 16. End For 17. i = i +2 18. $TG_{input} \leftarrow TG_{output}$ 19. End while 20. Return TG_{output}

i.e., particular configurations of relationships between entities. Generally, there are functionalities in existing querying solutions to extract subgraph structures of classic graphs. However, these subgraph structures do not include time dependent information. We therefore propose the $match_{pattern}$ operator to extract subgraph structures with time dependent information.

Definition 18. *A $match_{pattern}$ operator returns, for an input graph TG_{input}, an output subgraph TG_{output} matching a user-defined pattern P that describes a particular structure of relationships between entities to find in the temporal graph. It is denoted as follows:*

$$match_{pattern} : TG_{input} \times P \rightarrow TG_{output}$$

where a pattern $P = (l^{E_i}, l^{R_j}, l^{E_k}, ..., l^{R_m}, l^{E_n})$ is a sequence of entity labels linked by relationship labels. The algorithm of the execution of the operator is presented in Table 3.5.

Example 3. *Let us consider the TG of the running example (Section 3.1.2). Users want to find students who interacted with teachers in classes and socialized with other students at day 5. The analysis implies to find a specific pattern $P = (STUDENT, SOCIALIZE, STUDENT', ATTEND\_CLASS, TEACHER)$. We can then apply a $match_{pattern}$ operation:*

$$TG_1 = match_{pattern}(TG, (STUDENT, SOCIALIZE, STUDENT',$$

$$ATTEND\_CLASS, TEACHER))$$

The $match_{pattern}$ operator extracts the entities from the first label $STUDENT$ and then from the third label $STUDENT'$ of the user-defined pattern P. Then, it extracts the relationship states from the second label $SOCIALIZE$ between the two previous entity classes to construct a subgraph with a pattern $STUDENT - SOCIALIZE - STUDENT'$. It iterates the operation for the next triple - $STUDENT', ATTEND\_CLASS, TEACHER$ - on the subgraph obtained from the first operation, to construct a subgraph with the whole pattern $STUDENT - SOCIALIZE - STUDENT' - ATTEND\_CLASS - TEACHER$.

Then, there is a temporal condition ($valid\_time\, d\, day5$) assigned to two types of interactions of the students: the interactions with other students and the interactions with teachers. We therefore apply a $match_{predicates}$ operation taking the subgraph TG_1 resulting from the previous operation as input:

$$TG_2 = match_{predicates}(TG_1, \{SOCIALIZE.valid\_time\, d\, day5\ AND \\ ATTEND\_CLASS.valid\_time\, d\, day5\})$$

This operation filters the states of the relationship types $SOCIALIZE$ and $ATTEND\_CLASS$ according to the temporal condition. We obtain the subgraph TG_2 in Fig 3.14. We observe that only the states of $SOCIALIZE$ and $ATTEND\_CLASS$ relationships that satisfy the temporal condition are kept in the result.

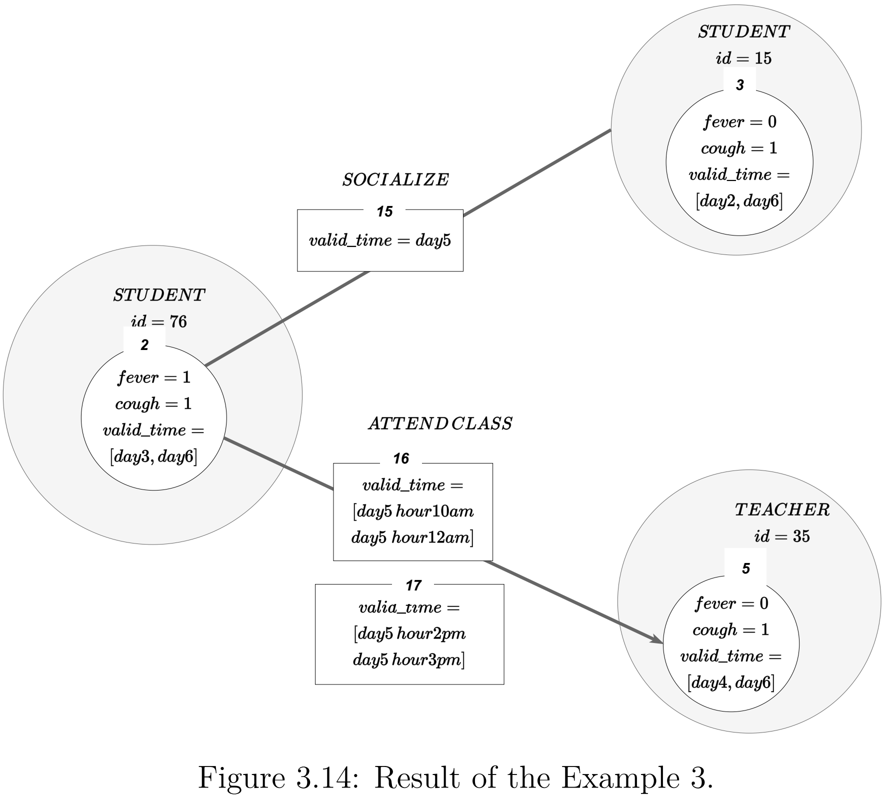


Figure 3.14: Result of the Example 3.

To conclude, the two operators we propose fully satisfy the desirable analysis criteria of a querying solution for TG (Section 2.4). The $match_{predicates}$ operator allows for manipulating the attributes of entities (or relationships), their attribute value as well as the time dimension of TG. Contrary to existing work, it allows fully manipulating the time dimension by integrating all possible Allen operators. The $match_{pattern}$ operator allows for manipulating the topology dimension of TG. Regarding the practicality criteria (Section 2.4), both operators provide high levels of user-orientation, composability and flexibility. TAs hey rely on business-oriented concepts (entities, relationships, states), users can express easily their queries in a way that directly relates to their domain without getting bogged down in implementation details. Both operators are composable by taking a TG as input and output. Since these operators are independent from implementation, they can be easily modified or be completed by supplementary operators to answer new user needs. The compatibility criteria of both operators is assessed at the logical and physical levels.

3.2 Logical Level

In this section, our objective is to satisfy the compatibility of our querying solution. It implies an implementation of our querying solution within existing technical environments and without major custom developments. To do so, we propose a logical level of our querying solution. The logical level provides a data-oriented framework to translate the conceptual operators into logical operators and independent to technical environments.

Most of current Graph Data Management Systems (GDBMS) support the property graph data model even if there is no standard specification of this data model (Angles, 2018). It can be considered as a logical data model, i.e., independent of a technical environement. Similarly, textual query languages of GDBMS relying on the property graph have common features. We propose to formalize these common features into logical operators manipulating the property graph model.

3.2.1 Temporal Graph to Property Graph

As a preliminary step, we have to map our conceptual temporal property graph TG into a logical property graph PG (Angles, 2018) to understand the concepts manipulated into the logical operators. To do so, as already discussed in Chapter 2, our conceptual model of temporal graph can be translated to the logical property graph as follows.

A property graph $PG = \langle V, D \rangle$ is composed of a set of vertices V and a set of edges D. Each edge is associated to a pair of vertices. Each vertex (or edge) can be labelled. Each property is a key-value pair. For each state s of an entity e in TG, a vertex is created in PG with a label corresponding to the label of e and a set of properties corresponding to the identifier of e, the attributes of s, the start and end instants of the valid time interval of s. An entity in the property graph corresponds therefore to a set of vertices having the same identifier. For each state s of a relationship r in TG, an edge is created in PG by connecting the two vertices corresponding to two states that r links, with a label corresponding to the label of r and a set of properties corresponding to the attributes of s, the start and end instants of the valid time interval of s. A relationship in the property graph corresponds therefore to a set of edges linking the same pair of vertices. The translation rules and transformation algorithm of our conceptual temporal graph into the logical property graph are detailed in Chapter 2 Section 5. We present the concepts we use for the transformation of our conceptual operators into the logical operators in Table 3.6.

Example 4. *Using the translation rules we define, we transform the TG in the running example in Fig 3.11 into the property graph in Fig 3.15.*

Temporal graph	**Property graph**
All entities E	all vertices V
All relationships R	all edges D
a state of an entity s	a vertex v
a state of a relationship s	an edge d
a label of an entity $l^{E'}$	a label of a vertex $l^{V'}$
a label of a relationship $l^{R'}$	a label of an edge $l^{D'}$
an attribute of an entity $a_q^{e_i}$	a property p
an entity's identifier id^{e_i}	a property p
an attribute of a relationship $a_d^{r_i}$	a property p
a valid time interval of an entity state T	two properties t_b and t_f^*
a valid time interval of a relationship state T	two properties t_b and t_f^*

Table 3.6: Transformation rules of our conceptual model into the logical model of property graph. $^*t_b$ is the start valid time instant of T and t_f is the ending valid time instant of T.

3.2.2 Logical Operators

The objective is to translate our conceptual operators into logical operators applicable to a property graph. To do so, we propose to formalize the three basic operators for manipulating property graphs found in the literature (Sharma et al., 2021; Angles et al., 2017). First and foremost, we have the basic graph pattern matching (bgpm) operator which enables to get subgraph that matches a user-defined pattern. Then, to obtain more complex patterns, the bgpm operator is extended with relational-like operators. We have the filter operator to restrict the matches of bgpm to user-defined conditions on properties and the projection operator to return user-defined output variables.

Definition 19. *A basic pattern matching operator consists of extracting a subgraph PG_{output} of an input property graph $PG_{input} = \langle V, D\rangle$ that matches a user-defined pattern $Q = (V', D')$.*

$$bgpm : PG_{input} \times Q \rightarrow PG_{output}$$

A pattern $Q = (V', D')$ is composed of a set of vertex variables $V' = \{x_1, ..., x_n\}$, where each x_i can be a vertex label, and a set of edge variables $D' = \{(x_1, y_1, x_2), ..., (x_{n-1}, y_n, x_n)\}$, where each y_i can be an edge label. A match h of $Q = (V', D')$ in $PG_{input} = \langle V, D\rangle$ is such that for each vertex $x_i \in V'$ and each edge $(x_i, y, x_j) \in D'$ we have $h(x_i) \in V$ and $h(x_i, y, x_j) \in D$.

Example 5. *Let us consider the property graph in Fig 3.15 denoted as PG. Users want to extract the interactions between students and teachers, they can use a basic pattern matching operator using the pattern $Q = (V', D')$ where $V' = \{STUDENT, TEACHER\}$ and $D' = \{(STUDENT, ATTEND\_CLASS, TEACHER)\}$. So we have:*

$$bgpm(PG, (\{STUDENT, TEACHER\}, \{(STUDENT, ATTEND\_CLASS, TEACHER)\})$$

This operation extracts the red subgraph with full lines from the property graph.

Definition 20. *A filter operator takes as input a graph PG_{input} following a pattern Q (i.e., the result of a bgpm operation) and selects the matches of the pattern that satisfy a user-defined filter F on a set of variables $X \subseteq Q$.*

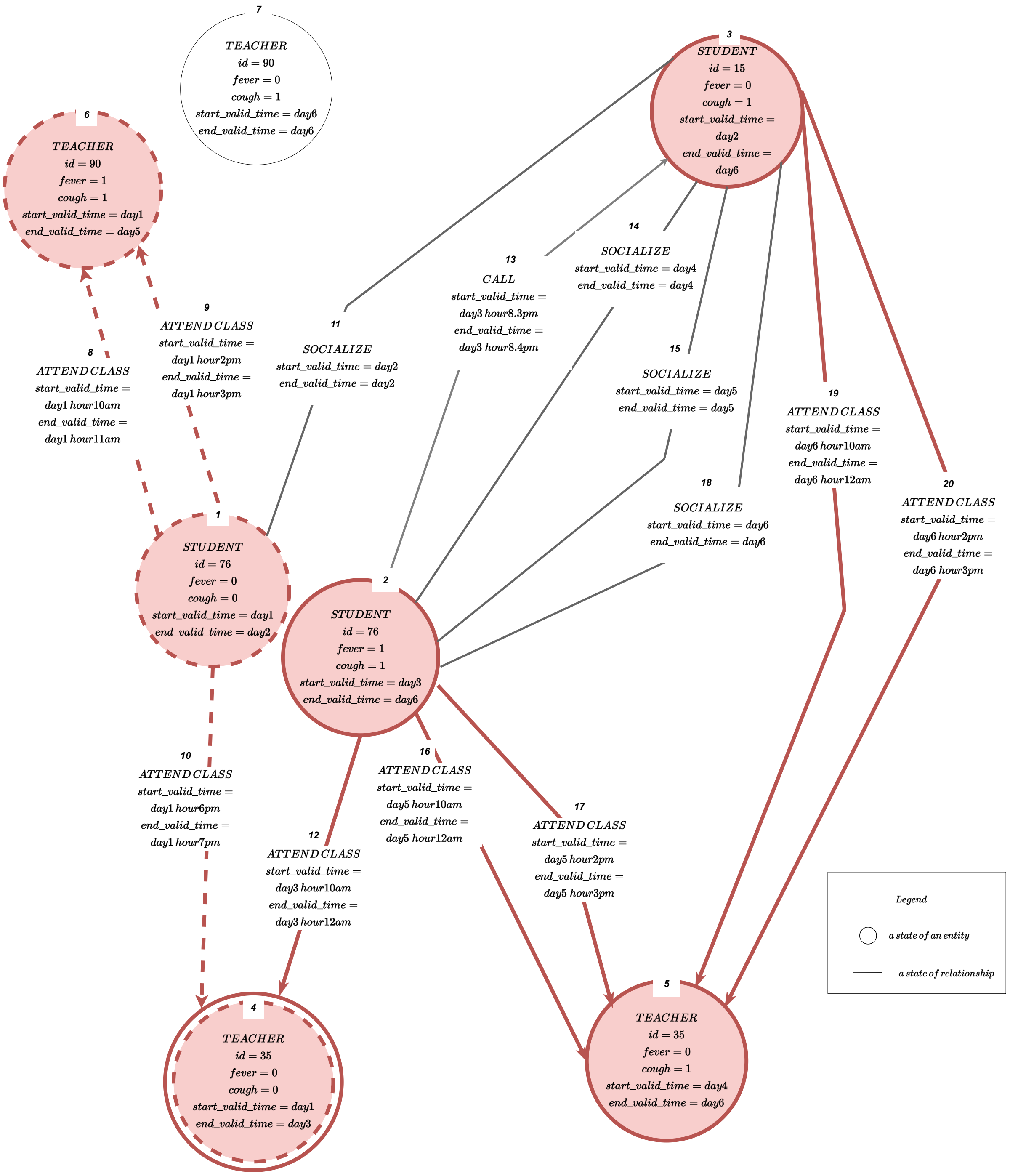


Figure 3.15: An example of property graph

$$filter : PG_{input} \times F(X) \rightarrow PG_{ouptut}$$

A filter $F(X) = \{x_1.f_1\beta...\beta x_n.f_n\}$ *is a combination of constraints, where each constraint* f_i *is assigned to a variable* $x_i \in X$ *and* β *is a boolean connector (e.g., AND, NOT, OR). A variable* x_i *can be either the whole set of vertices* V*, the whole set of edges* D*, a vertex label* $l^{V'}$ *or an edge label* $l^{D'}$*. A constraint* $f_i = p\,\theta\,w$ *is composed by a property* p*, a comparison operator* θ *and a user-defined value* w*. We denote* $M \in PG_{input}$ *the set of matches to* Q*. For each match* $m \in M$*, if* $F(X) = true$ *and* $X \subseteq Q$*, then* $PG_{output} = PG_{output} \cup m$*. After evaluating all matches, if* PG_{output} *is empty, it means that there are* *no matches that satisfy the filter.*

Example 6. *Let us consider the property graph in Fig 3.15. If users want to extract the interactions between students and teachers during day 1, they can use the filter operator using as input the bgpm operation in Example 5 and then apply the filter* $F(ATTEND\_ClASS) = \{ATTEND\_CLASS.start\,valid time \geq day\,1\,00 : 00am\,AND\,ATTEND\_CLASS.end\,valid time < day\,2\,00 : 00am\}$*. This operation results on the extraction of the red subgraph with dotted lines.*

Definition 21. *A projection operator takes as input the output graph of another operation* PG_{input} *and selects a subset of this input graph* PG_{ouptut} *according to user-defined selected output variables* O*.*

$$project : PG_{input} \times O \rightarrow PG_{ouptut}$$

An output graph can be selected properties of a vertex (or edge) label to only show the value of properties in the result, or the whole subgraph resulting from the previous operation. In other terms, for each vertex (or edge) in PG_{input}*, only the properties specified in the set* O *are retained, while all other properties are discarded.*

Example 7. *Let us consider the property graph in Fig 3.15. If users want to show the identifiers of all persons of the property graph, we can apply the projection operator using* $O = \{STUDENT.id, TEACHER.id\}$ *and taking as input the result of a bgpm operation (see Example 5).*

3.2.3 Mapping Conceptual Operators to Logical Operators

In this section, we define translation rules of our conceptual operators into logical operators. The algorithms of our proposed operators (Tables 3.3 and 3.5) define the scope of the sequence of logical operators.

The conceptual operator $match_{predicates}(TG, \pi)$ can be translated at the logical level through the execution of the filter operator $PG_1 = filter(PG, F(X))$ where the combination of predicates π is mapped to the filter $F(X)$. Then, the projection operator $PG_2 = project(PG_1, PG_1)$ enables to return the graph resulting from the previous operation. To do so, we propose a transformation process between the conceptual $match_{predicates}$ and logical operators via the Algorithm 2. It consists mainly in transforming the combination of predicates $\pi \in match_{predicates}(TG, \pi)$ into the filter $F(X) \in filter(PG, F(X))$ as the following:

1. Each connector $\beta \in \pi$ is the same as each connector $\beta \in F(X)$.

2. Each attribute predicate $\pi_i = X.a\,\theta\,w \in \pi$ is directly translated into each constraint $x_i.f_i \in F(X)$ such that $x_i = X$ and $f_i = p\,\theta\,w$. An attribute a in TG corresponds to a property in PG (Table 3.6).

3. Each temporal predicate $\pi_i = X.T\,\alpha\,T_u \in \pi$ can be mapped to two constraints $x_i.f_i \in F(X)$. Since the valid time interval T in TG corresponds to two properties t_b and t_f in PG (Table 3.6), it is translated to $x_i.t_b\theta w_1\,AND\,x_i.t_f\theta w_2$. The Allen operators α in π_i are translated to a comparison operator θ. To do so, the Table 3.2 describes Allen operators using comparison operators. The user-defined time inter-

val T_u is translated to two values w_1 and w_2 representing respectively the start valid time instant of T_u and the end valid time instant of T_u.

4. Each subject $X \in \pi_i$ corresponds to $x_i \in x_i.f_i$ such that:

   - if the subject is all entities in TG $(X = E)$ then $x_i = V$, i.e., all vertices in PG;

   - if the subject is all relationships R in TG $(X = R)$, then $x_i = D$, i.e., all edges in PG;

   - if the subject is all entities belonging to the class $l^{E'}$ in TG $(X = l^{E'})$, then $x_i = l^{V'} = l^{E'}$, i.e., to all vertices having the vertex label $l^{V'} = l^{E'}$ in PG;

   - if the subject is all relationships belonging to the type $l^{R'}$ in TG $(X = l^{R'})$, $x_i = l^{D'} = l^{R'}$, i.e., to all edges having the edge label $l^{D'} = l^{R'}$ in PG;

As a result of the Algorithm 2, we obtain translation rules of the conceptual operator $match_{predicates}(TG, \pi)$ into logical operators in Table 3.7.

The conceptual operator $match_{pattern}(TG, P)$ can be translated at the logical level through the execution of the basic pattern matching operator $bgpm(PG, Q)$ where the pattern P is mapped to the pattern Q. Then, it uses the projection operator as in the chain of operations of $match_{predicates}$ to get the result set. To do so, we propose a transformation process between the conceptual $match_{pattern}$ and logical operators via the Algorithm 3. It consists mainly in transformation of the pattern P of $match_{pattern}(TG, P)$ to the pattern Q of $bgpm(PG, Q)$ as the following:

1. Each entity label $l^{E'} \in P$ is translated to a vertex label $x_i = l^{V'} = l^{E'} \in Q$.

2. Each relationship label $l^{R'} \in P$ is translated to an edge label $x_i = l^{D} = l^{R'} \in Q$.

As a result of the Algorithm 3, we obtain translation rules of the conceptual operator $match_{pattern}(TG, P)$ into logical operations in Table 3.8. In conclusion, our two conceptual operators for temporal graph data have high compatibility with property graph-based environments. Indeed, they can be translated to property graph operators without difficulty using the mapping rules and algorithms we propose.

Algorithm 2: Mapping algorithm: from $match_{predicates}$ to logical property graph operators

```
Input: a Temporal Graph $TG = \langle E, R \rangle$ transformed into a Property Graph $PG = \langle V, D \rangle$, the
       $match_{predicates}(TG, \pi)$ operator, a user-defined combination of predicates $\pi_i$
Output: $PG_1 = bgpm(PG, Q)$
$PG_2 = filter(PG, F(X))$
$PG_3 = project(PG_2, PG_2)$
Create a filter $F(X)$
foreach $\pi_i \in \pi$ do
    if $\pi_i = X.a\,\theta\,w$ then
        Get a property $p \in PG$ a such that $p = a$
        if $X = E$ then
            Create $x_i.f_i = V.p\,\theta\,w$
            $F(X) = F(X) + \beta x_i.f_i$ where $\beta$ is preceeding $\pi_i$ in $\pi$
        else if $X = l^{E'}$ then
            Create $x_i.f_i = l^{V'}.p\,\theta\,w$ where $l^{V'} = l^{E'}$
            $F(X) = F(X) + \beta x_i.f_i$ where $\beta$ is preceeding $\pi_i$ in $\pi$
        else if $X = R$ then
            Create $x_i.f_i = D.p\,\theta\,w$
            $F(X) = F(X) + \beta x_i.f_i$ where $\beta$ is preceeding $\pi_i$ in $\pi$
        else if $X = l^{R'}$ then
            Create $x_i.f_i = l^{D'}.p\,\theta\,w$ where $l^{D'} = l^{R'}$
            $F(X) = F(X) + \beta x_i.f_i$ where $\beta$ is preceeding $\pi_i$ in $\pi$
    else if $\pi_i = X.T\,\alpha\,T_u$ then
        Get the property $t_b \in PG$ such that $t_b$ is the start valid time instant of $T \in TG$
        Get the property $t_f \in PG$ such that $t_f$ is the end valid time instant of $T \in TG$
        Create the value $w_1$ as the start valid time instant of $T_u$
        Create the value $w_2$ as the end valid time instant of $T_u$
        if $X = E$ then
            Create $x_i.f_i = V.t_b\,\theta\,w_1\,AND\,V.t_f\,\theta\,w_2$
            $F(X) = F(X) + \beta x_i.f_i$ where $\beta$ is preceeding $\pi_i$ in $\pi$
        else if $X = l^{E'}$ then
            Create $x_i.f_i = l^{V'}.t_b\,\theta\,w_1\,AND\,l^{V'}.t_f\,\theta\,w_2$ where $l^{V'} = l^{E'}$
            $F(X) = F(X) + \beta x_i.f_i$ where $\beta$ is preceeding $\pi_i$ in $\pi$
        else if $X = R$ then
            Create $x_i.f_i = D.t_b\,\theta\,w_1\,AND\,D.t_f\,\theta\,w_2$
            $F(X) = F(X) + \beta x_i.f_i$ where $\beta$ is preceeding $\pi_i$ in $\pi$
        else if $X = l^{R'}$ then
            Create $x_i.f_i = l^{D'}.t_b\,\theta\,w_1\,AND\,l^{D'}.t_f\,\theta\,w_2$ where $l^{D'} = l^{R'}$
            $F(X) = F(X) + \beta x_i.f_i$ where $\beta$ is preceeding $\pi_i$ in $\pi$
    Use $F(X)$ as input of a filter operation on $PG$
    Apply $PG_1 = bgpm(PG, Q)$
    Apply $PG_2 = filter(PG_1, F(X))$
    Create the output variables $O$
    $O \leftarrow PG_2$
    Apply $PG_3 = project(PG_2, O)$
```

	Conceptual Level	Logical Level
(1) a $match_{predicates}$ operation	$match_{predicates}(TG, \pi)$	$PG_1 = bgpm(PG, X)$ $PG_2 = filter(PG_1, F(X))$ $PG_3 = project(PG_2, PG_2)$
(2) a $match_{predicates}$ with a combination of predicates	$match_{predicates}(TG,$ $\{\pi_1\ \beta ... \beta\ \pi_n\})$	$PG_1 = bgpm(PG, X)$ $PG_2 = filter(PG_1, x_1.f_1 \beta ... \beta x_n.f_n)$ $PG_3 = project(PG_2, PG_2)$
(3) $match_{predicates}$ with an attribute predicate	$match_{predicates}(TG,$ $X.a\, \theta\, w)$	$PG_1 = bgpm(PG, x_i)$ $PG_2 = filter(PG, x_i.p\, \theta\, w)$ $PG_3 = project(PG_2, PG_2)$
(4) a $match_{predicates}$ with an attribute predicate on the subject E	$match_{predicates}(TG,$ $E.a\, \theta\, w)$	$PG_1 = bgpm(PG, V)$ $PG_2 = filter(PG_1, V.p\, \theta\, w)$ $PG_3 = project(PG_2, PG_2)$
(5) a $match_{predicates}$ with an attribute predicate on the subject $l^{E'}$	$match_{predicates}(TG,$ $l^{E'}.a\, \theta\, w$	$PG_1 = bgpm(PG, l^{V'})$ $PG_2 = filter(PG_1, l^{V'}.p\, \theta\, w)$ $PG_3 = project(PG_2, PG_2)$
(6) a $match_{predicates}$ with an attribute predicate on the subject R	$match_{predicates}(TG,$ $R.a\, \theta\, w)$	$PG_1 = bgpm(PG, D)$ $PG_2 = filter(PG_1, D.p\, \theta\, w)$ $PG_3 = project(PG_2, PG_2)$
(7) a $match_{predicates}$ with an attribute predicate on the subject $l^{R'}$	$match_{predicates}(TG,$ $l^{R'}.a\, \theta\, w)$	$PG_1 = bgpm(PG, l^{D'})$ $PG_2 = filter(PG_1, l^{D'}.p\, \theta\, w)$ $PG_3 = project(PG_2, PG_2)$
(8) $match_{predicates}$ with a temporal predicate	$match_{predicates}(TG,$ $X.T\, \alpha\, T_u)$	$PG_1 = bgpm(PG, x_i)$ $PG_2 = filter(PG_1, x_i.t_b \theta\, w_1$ AND $x_i.t_f \theta\, w_2)$ $PG_3 = project(PG_2, PG_2)$
(9) a $match_{predicates}$ with a temporal predicate on the subject E	$match_{predicates}(TG,$ $E.T\, \alpha\, T_u)$	$PG_1 = bgpm(PG, V)$ $PG_2 = filter(PG_1, V.t_b \theta\, w_1$ AND $V.t_f \theta\, w_2)$ $PG_3 = project(PG_3, \emptyset)$
(10) a $match_{predicates}$ with a temporal predicate on the subject $l^{E'}$	$match_{predicates}(TG,$ $l^{E'}.T\, \alpha\, T_u)$	$PG_1 = bgpm(PG, l^{V'})$ $PG_2 = filter(PG_1, l^{V'}.t_b \theta\, w_1$ AND $l^{V'}.t_f \theta\, w_2)$ $PG_3 = project(PG_2, PG_2)$
(11) a $match_{predicates}$ with a temporal predicate on the subject R	$match_{predicates}(TG,$ $R.T\, \alpha\, T_u)$	$PG_1 = bgpm(PG, D)$ $PG_2 = filter(PG_1, D.t_b \theta\, w_1$ AND $D.t_f \theta\, w_2)$ $PG_3 = project(PG_3, PG_3)$
(12) a $match_{predicates}$ with a temporal predicate on the subject $l^{R'}$	$match_{predicates}(TG,$ $l^{R'}.T\, \alpha\, T_u)$	$PG_1 = bgpm(PG, l^{D'})$ $PG_2 = filter(PG_1, l^{D'}.t_b \theta\, w_1$ AND $l^{D'}.t_f \theta\, w_2)$ $PG_3 = project(PG_2, PG_2)$

Table 3.7: Translation rules of $match_{predicates}$ operator

Algorithm 3: Mapping algorithm: from $match_{pattern}(TG, P)$ to logical property graph operations

```
Input: a Temporal Graph $TG = \langle E, R \rangle$ transformed into a Property Graph $PG = \langle V, D \rangle$
the $match_{pattern}(TG, P)$ operator, a user-defined pattern $P$
Output: $PG_1 = bgpm(PG, Q)$
$PG_2 = project(PG_1, PG_1)$
Create a pattern $Q = (V', D')$
foreach triple $(l^{E_i}, l^{R_j}, l^{E_k}) \in P$ do
    Get a vertex label $l^{V_i} \in PG$ such that $l^{V_i} = l^{E_i}$
    Get a vertex label $l^{V_k} \in PG$ such that $l^{V_k} = l^{E_k}$
    Get an edge label $l^{D_j} \in PG$ such that $l^{D_j} = l^{R_j}$
    $V' \leftarrow V' \cup \{l^{V_i}, l^{V_k}\}$
    $D' \leftarrow D' \cup \{(l^{V_i}, l^{D_j}, l^{V_k})\}$
Use $Q$ as input of a bgpm operation on $PG$
Apply $PG_1 = bgpm(PG, Q)$
Create the output variables $O$
$O \leftarrow PG_1$
Apply $PG_2 = project(PG_1, O)$
```

	Conceptual Level	Logical Level
(1) a $match_{pattern}$ operation	$matching_{pattern}(TG, P)$	$PG_1 = bgpm(PG, Q)$ $PG_2 = project(PG_1, PG_1)$
(2) a $match_{pattern}$ with a specific pattern P	$match_{pattern}(TG, (l^{E_i}, l^{R_j}, l^{E_k})$	$PG_1 = filter(PG, (\{l^{V_i}, l^{V_k}\}, \{(l^{V_i}, l^{D_j}, l^{V_k})\})$ $PG_2 = project(PG_1, PG_1)$

Table 3.8: Implementation of $match_{pattern}$ operator

3.3 Physical Level

In this section, we define translation rules of logical operators into some textual query languages (QL) of GDBMS supporting the property graph model. We choose the textual query languages of two GDBMS : (i) Cypher of Neo4j because it is widely used in the scientific community (Debrouvier et al., 2021; Cattuto et al., 2013) and in the industry (Tian, 2022), and (ii) OrientDB's QL because it is used in the projects of ACTIVUS Group.

Operators at the logical level are translated into a query written in a QL. A query in QL is made from several clauses chained together. The semantics of a query in QL are defined by the chain of its clauses.

In Neo4j and OrientDB's QL, the $MATCH$ clause allows specifying a pattern we search in a database. A basic graph pattern matching operation ($bgpm(PG, Q)$) at the logical level is then achieved by formulating a query in Neo4j and OrientDB's QL using the $MATCH$ clause followed by the pattern. The construction of such queries is detailed in Table 3.9.

In Neo4j and OrientDB's QL, the $WHERE$ clause is not a clause in its own right. It is part of the $MATCH$ clause. $WHERE$ adds constraints to the pattern described in $MATCH$. A filter operation $(filter(PG, F(x)))$ at the logical level is then achieved by formulating a query in Neo4j and OrientDB's QL using the $WHERE$ clause. The construction of such queries is detailed in Table 3.10.

In Neo4j and OrientDB's QL, the $RETURN$ clause defines the parts of a pattern (vertices, edges, and/or properties) to be included in the query result. It is one of the last clauses in a query. A projection operation $(project(PG, O))$ at the logical level is then achieved by formulating a query in Neo4j and OrientDB's QL using the $RETURN$ clause. The construction of such queries is detailed in Tables 3.9 and 3.10.

It is important to notice there are operators in Neo4j and OrientDB's QL that have the same semantics as comparison operators θ in predicates at the logical level. We can find more details of the semantics and syntax of Neo4j and OrientDB's QL respectively at `https://neo4j.com/docs/cypher-manual/current/syntax/` and `https://orientdb.com/docs/2.2.x/SQL-Match.html`.

We observe in Tables 3.9 and 3.10 that Neo4j and OrientDB's QL include sufficient functionalities to easily implement our conceptual operators. In other environments, our proposed operators could not be easily implementable into QL. Due to the rich functionalities of the Neo4j and OrientDB's QL, the mapping of our operators is direct and simple.

In conclusion, our two conceptual operators for temporal graph data can be actually implemented into textual query languages for the property graph model. This is possible because our conceptual operators are translatable directly to logical property graph operators in the Section 3.2.3. Therefore, our conceptual operators can be implemented into textual query languages for the property graph model (i.e, textual query languages embedding the functionalities of the logical property graph operators).

	Conceptual Level	Physical Level	
		Neo4j QL (Cypher)	OrientDB QL
(1) a $match_{pattern}$ operation	$matching_{pattern}(TG, P)$	$MATCH\, pattern$ $RETURN\, *$	$MATCH\{pattern\}$ $RETURN\, *$
(2) a $match_{pattern}$ with a specific pattern P	$match_{pattern}(TG,$ $(l^{E_i}, l^{R_j}, l^{E_k})$	$MATCH\,(object1 : l^{V_i}) - (object2 : l^{D_j}) - (object3 : l^{V_k})$ $RETURN\, *$	$MATCH\{class : l^{V_i}\}.outE(l^{D_j}).inV(l^{V_k})$ $RETURN\, *$

Table 3.9: Implementation of $match_{pattern}$ operator

	Conceptual Level	Physical Level	
		Cypher	OrientDB
(1) a $match_{predicates}$ operation	$match_{predicates}(TG, \pi)$	$MATCH pattern$ $WHERE\, constraints$ $RETURN *$	$MATCH\, \{pattern,$ $where: constraints\}$ $RETURN *$
(2) a $match_{predicates}$ with a combination of predicates	$match_{predicates}(TG,$ $\{\pi_1\ \beta..\beta\, \pi_n\})$	$MATCH(object)$ $WHERE\, constraint1\, \beta, ...\beta\, constraintN$ $RETURN *$	$MATCH\, \{object,$ $where: constraint1\, \beta, ...\beta\, constraintN\}$ $RETURN *$
(3) a $match_{predicates}$ with an attribute predicate	$match_{predicates}(TG,$ $X.a\, \theta\, w)$	$MATCH\, (object)$ $WHERE\, object.p\theta\, w$ $RETURN *$	$MATCH\, \{class: object,$ $where: (p\theta\, w)\}$ $RETURN *$
(4) a $match_{predicates}$ with an attribute predicate on the subject E	$match_{predicates}(TG,$ $E.a\, \theta\, w)$	$MATCH\, (n)$ $WHERE\, n.p\, \theta\, w$ $RETURN *$	$MATCH\, \{class: V,$ $where: p\, \theta\, w\}$ $RETURN *$
(5) a $match_{predicates}$ with an attribute predicate on the subject $l^{E'}$	$match_{predicates}(TG,$ $l^{E'}.a\, \theta\, w)$	$MATCH\, (n: l^{V'})$ $WHERE\, n.p\, \theta\, w$ $RETURN *$	$MATCH\, \{class: l^{V'},$ $where: p\, \theta\, w\}$ $RETURN *$
(6) a $match_{predicates}$ with an attribute predicate on the subject R	$match_{predicates}(TG,$ $R.a\, \theta\, w$	$MATCH() - [r] - ()$ $WHERE\, r.p\, \theta\, w$ $RETURN *$	$MATCH\, \{class: D$ $where: p\, \theta\, w\}$ $RETURN *$
(7) a $match_{predicates}$ with an attribute predicate on the subject $l^{R'}$	$match_{predicates}(TG,$ $l^{R'}.a\, \theta\, w)$	$MATCH() - [r: l^{D'}] - ()$ $WHERE\, r.p\, \theta\, w$ $RETURN *$	$MATCH\, \{class: l^{D'}$ $where: p\, \theta\, w\}$ $RETURN *$
(8) a $match_{predicates}$ with a temporal predicate	$match_{predicates}(TG,$ $X.T\, \alpha\, T_u)$	$MATCH\, (object)$ $WHERE\, object.t_b\theta\, w_1$ AND $object.t_f\theta\, w_2$ $RETURN *$	$MATCH\, \{class: object,$ $where: t_b\theta\, w_1$ AND $t_f\theta\, w_2\}$ $RETURN *$
(9) a $match_{predicates}$ with a temporal predicate on the subject E	$match_{predicates}(TG,$ $E.T\, \alpha\, T_u)$	$MATCH\, (n)$ $WHERE\, n.t_b\theta\, w_1$ AND $n.t_f\theta\, w_2$ $RETURN *$	$MATCH\, \{class: V,$ $where: t_b\theta\, w_1$ AND $t_f\theta\, w_2\}$ $RETURN *$
(10) a $match_{predicates}$ with a temporal predicate on the subject $l^{E'}$	$match_{predicates}(TG,$ $l^{E'}.T\, \alpha\, T_u)$	$MATCH\, (n; l^{V'})$ $WHERE\, n.t_b\theta\, w_1$ AND $n.t_f\theta\, w_2$ $RETURN *$	$MATCH\, \{class: l^{V'},$ $where: t_b\theta\, w_1$ AND $t_f\theta\, w_2\}$ $RETURN *$
(11) a $match_{predicates}$ with a temporal predicate on the subject R	$match_{predicates}(TG,$ $R.T\, \alpha\, T_u)$	$MATCH() - [r] - ()$ $WHERE\, r.t_b\theta\, w_1$ AND $r.t_f\theta\, w_2$ $RETURN *$	$MATCH\, \{class: D$ $where: t_b\theta\, w_1$ AND $t_f\theta\, w_2\}$ $RETURN *$
(12) a $match_{predicates}$ with a temporal predicate on the subject $l^{R'}$	$match_{predicates}(TG,$ $l^{R'}.T\, \alpha\, T_u)$	$MATCH() - [r: l^{D'}] - ()$ $WHERE\, r.t_b\theta\, w_1$ AND $r.t_f\theta\, w_2$ $RETURN *$	$MATCH\, \{class: l^{D'}$ $where: t_b\theta\, w_1$ AND $t_f\theta\, w_2\}$ $RETURN *$

Table 3.10: Implementation of $match_{predicates}$ operator

4 Experimental Assessments

We run experiments with the objective of validating the feasibility of our proposed operators through their implementation in technical environments.

4.1 Technical Environment

Our experiments are conducted on a PowerEdge R630, 16 CPUs x Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40Ghz, 63.91 GB. Two virtual machines are installed on this hardware. Each virtual machine has 6 GB in terms of RAM and 100 GB in terms of disk size. On each of the two virtual machines, we installed respectively a graph database compatible with the property graph model: (i) Neo4j (version 4.1.3) and (ii) OrientDB (version 3.0.4). To avoid any bias in the disk management, we did not use any customized optimization techniques, but relied on default tuning of Neo4j

Table 3.11: Characteristics of datasets. *Y= Yes, N= No, AV = Attribute Value, AS = Attribute Set, T = Topology.*

Implementation	**Social Experiment**	**Citibike**
Number of vertices	33 934	2 861
Number of edges	2 168 270	27 561 618
Evolution types of entities	AV, T	AV, AS, T
Evolution types of relationships	T	AV, AS, T

and OrientDB. The technical details of our experiments are available on the website `https://gitlab.com/2573869/queryingtemporalpropertygraphs`.

4.2 Datasets

We use the Social Experiment [7] and the Citibike [8] real datasets from our previous experiments. We can therefore find more details about those datasets in Section 7 of Chapter 2.

The objective of using data coming from different sources is to guarantee that our datasets cover different domains to formulate different analyses. On the one hand, the Social Experiment dataset is about epidemiological contagion among students of a dormitory. The dataset traces the changes in the students' symptoms (fever, cough, depression, etc.) and the different types of interactions they have over time (proximity, closefriend, socialize, etc.). On the other hand, the Citibike dataset is about biking trips in New York City using bikes of the Citibike company. The dataset traces the trips of users between stations with the characteristics of the trips (identifier of the bike, user type, birth year of the user, trip duration, start and end time of the trip) as well as the characteristics of stations (station name, latitude and longitude).

Both datasets were transformed into a temporal graph based on the translation rules defined in Section 4 of Chapter 2. Then, we store both datasets in Neo4j and OrientDB by applying the translation rules of the temporal graph into the property graph in Section 5 of Chapter 2. We can find in Table 3.11 the details of both implementations.

4.3 Benchmark Analyses

We define 7 types of analyses to be run on our datasets. They correspond to the types of analyses we can formulate over the different dimensions of datasets:

- (1) analysis including attribute dimension only;
- (2) analysis including time dimension only;
- (3) analysis including attribute and time dimensions;
- (4) analysis including topology dimension only;
- (5) analysis including topology and attribute dimensions;
- (6) analysis including topology and time dimensions;

[7] `http://realitycommons.media.mit.edu/socialevolution.html`

[8] `https://www.citibikenyc.com/system-data`

Table 3.12: Queries on the Social Experiment Dataset

	Query	Dimension
(1)	Which students had a fever or cough?	attribute
(2)	What is the health status of students at the day 2009-03-06?	time
(3)	Which students started to have a fever at the day 2009-03-06?	attribute, time
(4)	Which students are close friends and make social activities together?	topology
(5)	Which students had a fever and were close physically to other students that had a cough?	topology, attribute
(6)	Which students were close physically during the day 2009-03-06?	topology, time
(7)	Which students had a fever and were close physically to other students that had a cough during the day 2009-03-06?	topology, attribute, time

- (7) analysis including topology, attribute and time dimensions;

By crossing the different dimensions of datasets in analyses, this distributes possible query scenarios in a balanced way.

By using the previous types of analyses, we write business analyses, identified by 1 to 7, for the Social Experiment Dataset in Table 3.12 and for the Citibike Dataset in Table 3.13. To translate these business analyses into queries, we will apply the following process. For each analysis, queries will be written differently according to the dataset. We will therefore associate each analysis with the letter a) if it concerns the Social Experiment Dataset and the letter b) is it concerns the Citibike Dataset. For each analysis, we can implement it differently. We will therefore associate the number (i) for the implementation in Neo4j's query language and the number (ii) for the implementation in OrientDB's query language. Therefore, we will have a total of 28 queries ($7\,business\,analyses \times 2\,datasets \times 2\,implementations$) .

4.4 Experimental Results

In this section, we present the translation of the business analyses presented in the previous section from the conceptual level (conceptual operators) to the physical level (queries). To so, we apply the translation rules from the conceptual level to the logical level presented in Section 3.2 and the guidelines for translating from the logical level to the physical level in Section 3.3.

At the conceptual and logical levels, the business analyses and datasets are taken into account. At the physical level, the implementation into Neo4j and OrientDB graph database systems and their textual query languages are taken into account. Queries are represented following the method presented in the previous section. For example, if the first business analysis of the Social Experiment is implemented in the Neo4j's query language, it will be referred as 1.a).(i).

Table 3.13: Queries on the Citibike Dataset

	Query	Dimension
(1)	What are the geographical coordinates of the station of Clinton St & GrandSt?	attribute
(2)	Retrieve trips during the day 2021-01-28	time
(3)	Retrieve trips during the day 2021-01-28 that are made by subscribers	attribute, time
(4)	Retrieve triplets of stations linked by trips	topology
(5)	Retrieve the trips of bikes (i.e., the identifiers of bikes is the same accross trips) passing through three stations and starting at the station '1 Ave & E 30 St '	topology, attribute
(6)	Retrieve triplets of stations linked by trips during the day 2021-01-28	topology, time
(7)	Retrieve the trips of bikes passing through three stations starting at the station '1 Ave & E 30 St ' and starting at the day 2021-01-28.	topology, attribute, time

4.4.1 Analyses including Attribute Dimension

For the business analyses numbered 1, including attribute dimension only, we have the following translation process:

At the conceptual level:
1.a) $TG_1 = match_{predicates}(TG, \{STUDENT.(fever = 1)\, OR\, STUDENT.cough = 1)$
1.b) $TG_1 = match_{predicates}(TG, \{STATION.stationname =' Clinton\, St\, \&\, Grand\, St'\})$

At the logical level:
1.a) $PG_1 = bgpm(PG, (\{STUDENT\}, \{\})$
$PG_2 = filter(PG_1, STUDENT.fever = 1\, OR\, STUDENT.cough)$
$PG_3 = project(PG_2, PG_2)$
1.b) $PG_1 = bgpm(PG, (\{STATION\}, \{\})$
$PG_2 = filter(PG_1, STATION.stationname =' Clinton\, St\, \&\, Grand\, St')$
$PG_3 = project(PG_2, PG_2)$

At the physical level:
1.a).(i) $MATCH(s : Student)\, WHERE\, s.fever =' 1'\, OR\, s.cough =' 1'\, RETURN*$
1.a).(ii) $MATCH\{class : Student, where : (fever =' 1')\, OR\, (cough =' 1')\}RETURN*$
1.b).(i) $MATCH(s : Station)\, WHERE\, s.stationname =' Clinton\, St\, \&\, Grand\, St'$ $RETURN*$
1.b).(ii) $MATCH\{class : Station, where : (stationname =' Clinton\, St\, \&\, Grand\, St')\}$ $RETURN*$

4.4.2 Analyses including Time Dimension

For the business analyses numbered 2, including time dimension only, we have the following translation process:

At the conceptual level:
2.a) $TG_1 = match_{predicates}(TG, \{STUDENT.T \circ [2009-03-06, 2009, 2009-03-07[\})$
2.b) $TG_1 = match_{predicates}(TG, \{TRIP.T\, d\, [2021-01-28, 2021-01-29[\})$

At the logical level:
2.a)$PG_1 = bgpm(PG, (\{STUDENT\}, \{\})$
$PG_2 = filter(PG_1, STUDENT.startvalidtime < 2009-03-07\, AND$
$STUDENT.endvalidtime \geq 2009-03-06)$
$PG_3 = project(PG_2, PG_2)$
2.b) $PG_1 = bgpm(PG, (\{TRIP\}, \{\})$
$PG_2 = filter(PG_1, TRIP.startvalidtime \geq 2021-01-28\, AND$
$TRIP.endvalidtime < 2021-01-29)$
$PG_3 = project(PG_2, PG_2)$

At the physical level:
2.a).(i) $MATCH(s : Student)$
$WHERE\, datetime(s.startvalidtime) < datetime('2009-03-07')$
$AND\, datetime(s.endvalidtime) \geq datetime('2009-03-06')\, RETURN*$
2.a).(ii) $MATCH\{class : Student, where : (startvalidtime.asdatetime().format('yyyy-MM-dd') < datetime('2009-03-07','yyyy-MM-dd').format('yyyy-MM-dd')\, AND\, endvalidtime.asdatetime().format('yyyy-MM-dd') >= datetime('2009-03-06','yyyy-MM-dd').format('yyyy-MM-dd')\}\, RETURN*$
2.b).(i) $MATCH(t : TRIP)\, WHERE\, datetime(t.startvalidtime) \geq datetime('2021-01-28')$
$AND\, datetime(t.endvalidtime) < datetime('2021-01-29')$
$RETURN*$
2.b).(ii)$MATCH\{class : Trip, where : (startvalidtime.asDate().format('yyyy-MM-dd' \geq date('2021-01-28','yyyy-MM-dd').format('yyyy-MM-dd')\, AND$
$(endvalidtime.asDate().format('yyyy-MM-dd')$
$>= date('2021-01-29','yyyy-MM-dd').format('yyyy-MM-dd')\}$
$RETURN*$

4.4.3 Analyses including Attribute and Time Dimensions

For the business analyses numbered 3, including attribute and time dimensions, we have the following translation process:

At the conceptual level:
3.a) $TG_1 = match_{predicates}(TG, \{STUDENT.fever = 1\, AND\, STUDENT.T\, s\, [2009-03-06, 2009-03-07[)\})$
3.b) $TG_1 = match_{predicates}(TG,$
$\{TRIP.T\, d\, [2021-01-28, 2021-01-29[\, AND$
$TRIP.usertype =' Subscriber'\})$

At the logical level:
3.a)$PG_1 = bgpm(PG, (\{STUDENT\}, \{\})$
$PG_2 = filter(PG_1, STUDENT.fever = 1\, AND\, STUDENT.startvalidtime = 2009 - 03 - 06)$
$PG_3 = project(PG_2, PG_2)$
3.b) $PG_1 = bgpm(PG, (\{TRIP\}, \{\})$
$PG_2 = filter(PG_1, TRIP.startvalidtime \geq 2021 - 01 - 28\, AND\, TRIP.endvalidtime < 2021 - 01 - 29\, AND\, TRIP.usertype =' Subscriber')$
$PG_3 = project(PG_2, PG_2)$

At the physical level:
3.a).(i) $MATCH(s : Student)\, WHERE fever = 1\, AND\, datetime(s.startvalidtime) = datetime('2009 - 03 - 06')\, RETURN*$
3.a).(ii) $MATCH\{class : Student, where : (fever =' 1')\, AND$
$(startvalidtime.asdatetime().format('yyyy - MM - dd') = datetime('2009 - 03 - 06',' yyyy - MM - dd').format('yyyy - MM - dd')\} RETURN*$
3.b).(i) $MATCH(t : TRIP)\, WHERE\, date(t.startvalidtime) \geq date('2021 - 01 - 28')\, AND\, time(t.endvalidtime) < time('2021 - 01 - 29')\, AND\, t.usertype =' Subscriber'$
$RETURN*$
3.b).(ii) $MATCH\{class : Trip, where : (startvalidtime.asDate().format('yyyy - MM - dd') \geq date('2021 - 01 - 28',' yyyy - MM - dd').format('yyyy - MM - dd')\, AND$
$(endvalidtime.asDate().format('yyyy - MM - dd')$
$>= date('2021 - 01 - 29',' yyyy - MM - dd').format('yyyy - MM - dd')$
$AND\, (usertype =' Substriber')\}$
$RETURN*$

4.4.4 Analyses including Topology Dimension

For the business analyses numbered 4, including topology dimension only, we have the following translation process:

At the conceptual level:
4.a) $TG_1 = match_{pattern}(TG, (STUDENT_1, CLOSEFRIEND, STUDENT_2,$
$SOCIALIZE, STUDENT_1))$
4.b) $TG_1 = match_{pattern}(TG, (STATION_1, TRIP_1, STATION_2, TRIP_2, STATION_3))$

At the logical level:
4.a) $PG_1 = bgpm(PG, (\{STUDENT_1, STUDENT_2\},$
$\{(STUDENT_1, CLOSEFRIEND, STUDENT_2),$
$(STUDENT_1, SOCIALIZE, STUDENT_2)\})$
$PG_2 = project(PG_1, PG_1)$
4.b) $PG_1 = bgpm(PG, (\{STATION_1, STATION_2, STATION_3\}, \{(STATION_1, TRIP_1, STATION_2)(STATION_2, TRIP_2, STATION_3)\})$
$PG_2 = project(PG_1, PG_1)$

At the physical level:

4.a).(i) $MATCH(s1 : Student)-[c : Closefriend]-(s2 : Student)-[s : Socialize]-(s1 : Student)\ RETURN*$
4.a).(ii) $MATCH\{class : Student, as : s1\}.outE(CloseFriend).inV(Student)\{as : s2\}$
$.outE(Socialize).inV(s1)$
$RETURN*$
4.b).(i) $MATCH(s1 : Station) - [t1 : Trip] - (s2 : Station) - [t2 : Trip] - (s3 : Station)\ RETURN*$
4.b).(ii) $MATCH\{class : Station, as : s1\}.outE(Trip)$
$.inV(Station)\{as : s2\}.outE(Trip).inV(Station)\{as : s3\}$
$RETURN*$

4.4.5 Analyses including Topology and Attribute Dimensions

For the business analyses numbered 5, including topology and attribute dimensions, we have the following translation process:

At the conceptual level:
5.a) $TG_1 = match_{predicates}(TG, \{STUDENT_1.(fever = 1)\ AND\ STUDENT_2.(cough = 1)\})$
$TG_2 = match_{pattern}(TG_1, (STUDENT_1, PROXIMITY, STUDENT_2))$
5.b) $TG_1 = match_{predicates}(TG, \{STATION_1.stationname =' 1\ Ave\ \&\ E\ 30\ St\ AND$
$TRIP_1.bikeid = TRIP_2.bikeid\})$
$TG_2 = match_{pattern}(TG, (STATION_1, TRIP_1, STATION_2, TRIP_2, STATION_3))$

At the logical level:
5.a)$PG_1 = bgpm(PG, \{STUDENT_1, STUDENT_2\}, \{PROXIMITY\})$
$PG_2 = filter(PG_1, STUDENT_1.fever = 1\ AND\ STUDENT_2.cough = 1)$
$PG_3 = project(PG_2, PG_2)$
5.b) $PG_1 = bgpm(PG, (\{STATION_1, STATION_2, STATION_3\},$
$\{(STATION_1, TRIP_1, STATION_2), (STATION_2, TRIP_2, STATION_3)\})$
$PG_2 = filter(PG_1, STATION.stationname =' 1\ Ave\ \&\ E, 30\ St'\ AND$
$TRIP_1.bikeid = TRIP_2.bikeid)$
$PG_3 = project(PG_2, PG_2)$

At the physical level:
5.a).(i) $MATCH(s1 : Student) - [p : Proximity] - (s2 : Student)\ WHERE\ s1.fever = 1\ AND\ s2.cough = 0$
$RETURN*$
5.a).(ii) $MATCH\{class : Student, as : s1, where : fever = 1\}.outE(Proximity)$
$.inV(Student)\{as : s2, where : cough = 1\})\ RETURN*$
5.b).(i) $MATCH(s1 : Station) - [t1 : Trip]- > (s2 : Station) - [t2 : Trip]- > (s3 : Station)$
$WHERE\ s1.stationname =' 1\ Ave\ \&\ E, 30\ St'\ AND\ t1.bikeid = t2.bikeid$
$RETURN*$
5.b).(ii) $MATCH\{class : Station, as : s1, where : (stationname =' 1\ Ave\ \&\ E, 30\ St')\}$
$.outE(Trip)\{as : t1\}.inV(Station)\{as : s2\}$
$.outE(Trip)\{as : t2, where : (bikeid = \$matched.(t1.bikeid)\}.inV(Station)\{as : s3\}$

$RETURN*$

4.4.6 Analyses including Topology and Time Dimensions

For the business analyses numbered 6, including topology and time dimensions, we have the following translation process:

At the conceptual level:
6.a) $TG_1 = match_{predicates}(TG, \{PROXIMITY.(T\,d\,[2009-03-06, 2009-03-07[)\})$
$TG_2 = match_{pattern}(TG_1, (STUDENT_1, PROXIMITY, STUDENT_2)$
6.b) $TG_1 = match_{predicates}(TG, \{TRIP_1.T\,d\,[2021-01-28, 2021-01-29[\,AND$
$TRIP_2.T\,d\,[2021-01-28, 2021-01-29[)\}$
$TG_2 = match_{pattern}(TG, (STATION_1, TRIP_1, STATION_2, TRIP_2, STATION_3))$

At the logical level:
6.a) $PG_1 = bgpm(PG, \{STUDENT_1, STUDENT_2\}, \{PROXIMITY\})$
$PG_2 = filter(PG_1, PROXIMITY.startvalidtime \geq 2009-03-06\,AND$
$PROXIMITY.endvalidtime < 2009-03-07$
$PG_3 = project(PG_2, PG_2)$
6.b) $PG_1 = bgpm(PG, (\{STATION_1, STATION_2, STATION_3\},$
$\{(STATION_1, TRIP_1, STATION_2),\ (STATION_2, TRIP_2, STATION_3)\})$
$PG_2 = filter(PG_1, TRIP.startvalidtime \geq 2021-01-28\,AND$
$TRIP.endvalidtime < 2021-01-29)$
$PG_3 = project(PG_2, PG_2)$

At the physical level:
6.a).(i) $MATCH(s1 : Student) - [p : Proximity] - (s2 : Student)$
$WHEREdatetime(p.startvalidtime) \geq datetime('2009-03-06')\,AND$
$datetime(p.endvalidtime) < datetime('2009-03-07')$
$RETURN*$
6.a).(ii) $MATCH\{class : Student, as : s1\}.outE(Proximity)\{as : p, where :$
$(startvalidtime.asdatetime().format('yyyy-MM-dd') \geq datetime('2009-03-06','yyyy-MM-dd').format('yyyy-MM-dd')\,AND\,(endvalidtime.asdatetime()$
$.format('yyyy-MM-dd') < datetime('2009-03-07','yyyy-MM-dd').format('yyyy-MM-dd')\}.inV(Student)\{as : s2\}$
$RETURN*$
6.b).(i) $MATCH(s1 : Station) - [t1 : Trip]- > (s2 : Station) - [t2 : Trip]- > (s3 : Station)$
$WHEREdatetime(t.startvalidtime) \geq datetime('2021-01-28')\,AND$
$datetime(t.endvalidtime) < datetime('2021-01-29')$
$RETURN*$
6.b).(ii) $MATCH\{class : Station, as : s1\}$
$.outE(Trip)\{as : t1, where : (startvalidtime.asDate().format('yyyy-MM-dd'$
$\geq date('2021-01-28','yyyy-MM-dd').format('yyyy-MM-dd')\,AND$
$(endvalidtime.asDate().format('yyyy-MM-dd' >= date('2021-01-29','yyyy-MM-dd').format('yyyy-MM-dd')\}.inV(Station)\{as : s2\}$
$.outE(Trip)\{as : t2, where : (startvalidtime.asDate().format('yyyy-MM-dd'$

$\geq date('2021-01-28','yyyy-MM-dd').format('yyyy-MM-dd')\, AND$
$(endvalidtime.asDate().format('yyyy-MM-dd'$
$>= date('2021-01-29','yyyy-MM-dd').format('yyyy-MM-dd')\}.inV(Station)\{as:s3\}$
$RETURN*$

4.4.7 Analyses including Topology, Attribute and Time Dimensions

For the business analyses numbered 7 including topology, attribute and time dimensions, we have the following translation process:

At the conceptual level:
7.a) $TG_1 = match_{predicates}(TG, \{\{STUDENT_1.(fever = 1)\, AND\, STUDENT_2.(cough = 1)\}\, AND\, PROXIMITY.(T\, d\, [2009-03-06, 2009-03-07[)\})$
$TG_2 = match_{pattern}(TG_1, (STUDENT_1, PROXIMITY, STUDENT_2)$
7.b) $TG_1 = match_{predicates}(TG, \{STATION_1.stationname =' 1\, Ave\, \&\, E\, 30\, St\, AND$
$TRIP_1.bikeid = TRIP_2.bikeid\})\, AND$
$TRIP_1.T\, s\, [2021-01-28, 2021-01-29$
$TG_2 = match_{pattern}(TG, (STATION_1, TRIP_1, STATION_2, TRIP_2, STATION_3))$

At the logical level:
7.a) $PG_1 = bgpm(PG, \{STUDENT_1, STUDENT_2\}, \{PROXIMITY\})$
$PG_2 = filter(PG_1, STUDENT_1.fever = 1\, AND$
$STUDENT_2.cough = 1\, AND\, PROXIMITY.startvalidtime \geq 2009-03-06\, AND$
$PROXIMITY.endvalidtime < 2009-03-07)$
$PG_3 = project(PG_2, PG_2)$
7.b) $PG_1 = bgpm(PG, (\{STATION_1, STATION_2, STATION_3\},$
$\{(STATION_1, TRIP_1, STATION_2), (STATION_2, TRIP_2, STATION_3)\})$
$PG_2 = filter(PG_1, STATION.stationname =' 1\, Ave\, \&\, E, 30\, St'\, AND\, TRIP_1.bikeid = TRIP_2.bikeid\, AND\, TRIP_1.startvalidtime = 2021-01-28)$
$PG_3 = project(PG_2, PG_2)$

At the physical level:
7.a).(i) $MATCH(s1: Student)-[p: Proximity]-(s2: Student)$
$WHEREs1.fever = 1\, AND\, s2.cough = 0$
$ANDdatetime(p.startvalidtime) \geq datetime('2009-03-06')$
$AND\, datetime(p.endvalidtime) < datetime('2009-03-07')$
$RETURN*$
7.a).(ii) $MATCH\{class: Student, as: s1, where: fever = 1\}.outE(Proximity)$
$\{as: p, where: (startvalidtime.asdatetime().format('yyyy-MM-dd')$
$\geq datetime('2009-03-06','yyyy-MM-dd').format('yyyy-MM-dd')\, AND$
$(endvalidtime.asdatetime().format('yyyy-MM-dd')$
$< datetime('2009-03-07','yyyy-MM-dd').format('yyyy-MM-dd')\}$
$.inV(Student)\{as: s2, where: cough = 1\}$
$RETURN*$
7.b).(i) $MATCH(s1: Station)-[t1: Trip]->(s2: Station)$

$-[t2 : Trip]- > (s3 : Station)$
$WHERE\, s1.stationname =' 1\, Ave\, \&\, E, 30\, St'' \, AND$
$t1.bikeid = t2.bikeid\, AND\, date(datetime(t1.startvalidtime)) = date('2021 - 01 - 28')$
$RETURN*$
7.b).(ii) $MATCH\{class : Station, as : s1, where : (stationname =' 1\, Ave\, \&\, E, 30\, St')\}$
$.outE(Trip)\{as : t1, where : (startvalidtime.asDate().format('yyyy - MM - dd' = date('2021 - 01 - 28',' yyyy - MM - dd').format('yyyy - MM - dd')\}$
$.inV(Station)\{as : s2\}$
$.outE(Trip)\{as : t2, where : (bikeid = \$matched.(t1.bikeid)\}.inV(Station)\{as : s3\}$
$RETURN*$

4.4.8 Summary

We observe that the translation of business analyses expressed in conceptual operations into queries expressed in Neo4j and OrientDB's QL is direct for both datasets. This is possible thanks to the mapping rules we propose at the logical level. No specific developments are necessary. It is important to notice that we use specific functionalities of Neo4j and OrientDB's QL such as date functions.

5 Conclusion

The *Temporal Graph* model we proposed in the previous chapter provides a way to represent the multidimensional aspects of graph data with temporal evolution: topology, attributes (of vertices and relationships) and time (evolution of topology and attributes). Some research works propose querying solutions to manipulate the multiple dimensions of temporal graph data in order to find information answering business questions of users. However, these querying solutions present some limitations regarding their analysis and practicality capabilities. On the one hand, some querying solutions cannot fully manipulate all dimensions of temporal graph data. They do not allow manipulating some information of the attribute dimension (attributes of edges in RDF-based temporal graph models) or time dimension (some Allen operators are not included). On the other hand, they lack of a generic framework which is independent of implementation environments. This limits their practicality aspects, i.e., user-orientation, composability, flexibility and compatibility.

Therefore, we have proposed in this chapter a complete querying solution, composed of three abstraction levels - conceptual (generic framework independent of implementation environmnent), logical (implementation framework) and physical (selection of technical environment) - to ensure that all analysis and practicality capabilities are fulfilled. At the conceptual level, we proposed fundamental operators to query temporal graph data that cover (i) basic operations on the topology and attribute dimensions and (ii) temporal dependent operations by crossing both topology and attribute dimensions with time dimension. Contrary to existing querying solutions, our operators enable to fully manipulate all granularities of the different dimensions. Moreover, they rely on business-oriented concepts from our Temporal Graph model instead of data-oriented concepts. This makes it easier for users to express queries in a way that aligns with their domain of expertise without worrying about implementation details. Then, our operators can be easily combined to express complex queries by taking a temporal graph as input and output. The previous features of our operators allow easy modification or addition of new querying functionalities to address new users' needs. In summary, the conceptual level of our querying solution offers several advantages over existing work. It completes the analysis capabilities for each dimension of temporal graph data and enhance the practicality aspects by providing high user-orientation, composability and flexibility.

We have proposed both a logical and physical level, providing an implementation framework for our conceptual operators. First, we have defined mapping rules to translate our conceptual operators directly into operators for querying the property graph model. This guarantees the compatibility of our conceptual operators to several technical environments supporting the property graph model. Then, we have discussed some implementation practices to translate the property graph operators directly into the textual query languages of Neo4j and OrientDB graph database management systems.

We have carried out experiments with real-world datasets coming from different sources and domains. We have proposed benchmark analyses covering different scenarios, starting from the manipulation of a single dimension of temporal graph data to the manipulation of the different dimensions (topology, attribute, time). From the results, we observe that the business analyses expressed via our conceptual operators are directly implementable (i.e., without specific developments) into the textual graph query languages of Neo4j and OrientDB via the mapping rules defined at the logical level.

Chapter 4

Knowledge Discovery in Temporal Graphs

Contents

1 Introduction

1.1 Context

In the previous chapter, we have proposed exploring *Temporal Graphs* (TG) using queries. Queries were means to extract descriptive information from TG, answering business questions related to '*What*', '*Who*', '*Where*' and '*When*' (Bellinger et al., 2004; Rowley, 2007). However, queries are not sufficient to answer '*How'* questions that delve into understanding the mechanisms, processes or dynamics behind certain phenomena or events (e.g., disease spreading) that occur and evolve within TG contexts. This understanding requires combining several pieces of information (e.g., different factors contributing to the phenomena) while queries extract isolated pieces of information from TG.

In this chapter, we address the problem of '*Knowledge Discovery*' in TG, i.e, discovering hidden information in TG answering '*How'* questions. More precisely, knowledge refers to the *combination of information pieces* which is interpreted (give a meaning) and evaluated (determine their value) to gain insights for understanding phenomena and making informed actions based on this understanding (Bellinger et al., 2004; Rowley, 2007). The complexity of discovering knowledge in TG lies in the need to combine information pieces of TG stemming from the diverse types of evolution, including the evolution of topology, attribute set, or attribute value (Chapter 2).

We choose the analytical approach of '*Pattern Mining*' in TG to discover knowledge in TG. Pattern Mining is a research field dealing with the challenges of handling various types of evolution within TG to discover knowledge. Specifically, Pattern Mining in TG refers to the process of extracting patterns using algorithms, with patterns representing hidden evolution mechanisms (Fournier-Viger et al., 2020a). Knowledge Discovery in TG can be illustrated in Fig 4.1 with Pattern Mining at its centre. In the following section, we will discuss, in more detail, the challenges of Pattern Mining in TG.

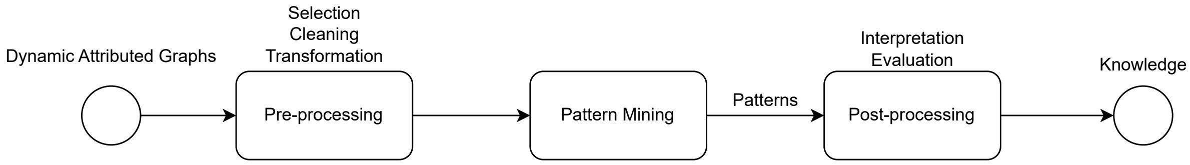


Figure 4.1: The process of Knowledge Discovery

1.2 Challenges

The primary challenge of Pattern Mining in TG is to design a pattern. This consists in constructing the structure of the pattern, which is intricately linked to the characteristics of an evolution mechanism being targeted. Consider the case of an epidemic in a population. We know that the evolution mechanism behind an epidemic is related to the sequence of changes in the health status of people (i.e, vertex attributes in the graph) and in the interactions between people (i.e., edges in the graph). A pattern should therefore integrate several aspects: information about evolution of attributes and topology and sequential dependencies between events that lead to the phenomenon.

Today, patterns include at best the evolution of topology and evolution of vertex attributes (Fournier-Viger et al., 2020a). However, they do not capture completely the information from the evolution of topology and the evolution of vertex attributes. For instance, they capture only the addition of edges over time in topology evolution or the same trends over time in vertex attribute values evolution. This limits the explanatory power of patterns for understanding evolution mechanisms (Desmier et al., 2013; Kaytoue et al., 2014; Fournier-Viger et al., 2019, 2020b).

Moreover, current patterns are designed to capture local patterns. They focus on the evolution within a specific group of connected vertices. While these patterns provide insights about localized behaviours, they might not effectively capture broader trends that span multiple groups (Desmier et al., 2013; Kaytoue et al., 2014; Cheng et al., 2017; Fournier-Viger et al., 2019, 2020b). For example, in the study of an epidemic, existing patterns may reveal how a virus propagates in a specific group of people. However, if this specific group is special (for example, they have innate resistance to this virus, or they were vaccinated), existing patterns lose general representativeness (i.e., capability of reflecting several groups of connected vertices) and cannot provide meaningful analysis for virus transmission. Indeed, understanding how phenomena propagate across different groups is crucial for making informed decisions. **In a nutshell, a complete and representative pattern is needed to provide more useful information.**

The second challenge of Pattern Mining in TG is to design an algorithm to mine a novel pattern. Existing mining algorithms are indeed designed with specific patterns in mind (Fournier-Viger et al., 2020a). They are not suitable for mining a novel pattern. However, Pattern Mining in TG is generally quite computationally expensive (time and memory) (Fournier-Viger et al., 2020a). Indeed, it requires exploring the different dimensions of TG (topology, attributes, time) but also verifying some constraints (e.g., frequency of appearance of patterns) to find subgraphs matching the patterns. **It is therefore essential to design an algorithm incorporating strategies to reduce the computational complexity**.

1.3 Contributions and Outline

Facing the previous challenges, we will present the following contributions in the remainder of the chapter: (i) a state of the art of existing patterns and mining algorithms to highlight their limitations (in Section 2), (ii) the definition of a novel pattern and a new mining problem (in Section 3), (iii) the design of an algorithm for our novel pattern with the proposition of a strategy to reduce computational complexity (in Section 4) and, (iv) experiments to extract our proposed pattern from real datasets using our algorithm (in Section 5).

2 Related Work

The research field of Pattern Mining in TG results from the extensions of the Sequential Pattern Mining field. The field of Sequential Pattern Mining addresses the extraction of knowledge in sequence databases. The Sequential Pattern Mining field has been built upon the field of Frequent Itemsets and Association Rules Mining. The Frequent Itemsets and Association Rules Mining field considers the extraction of knowledge in databases. In the following section, we therefore present the fields of Frequent Itemsets and Association

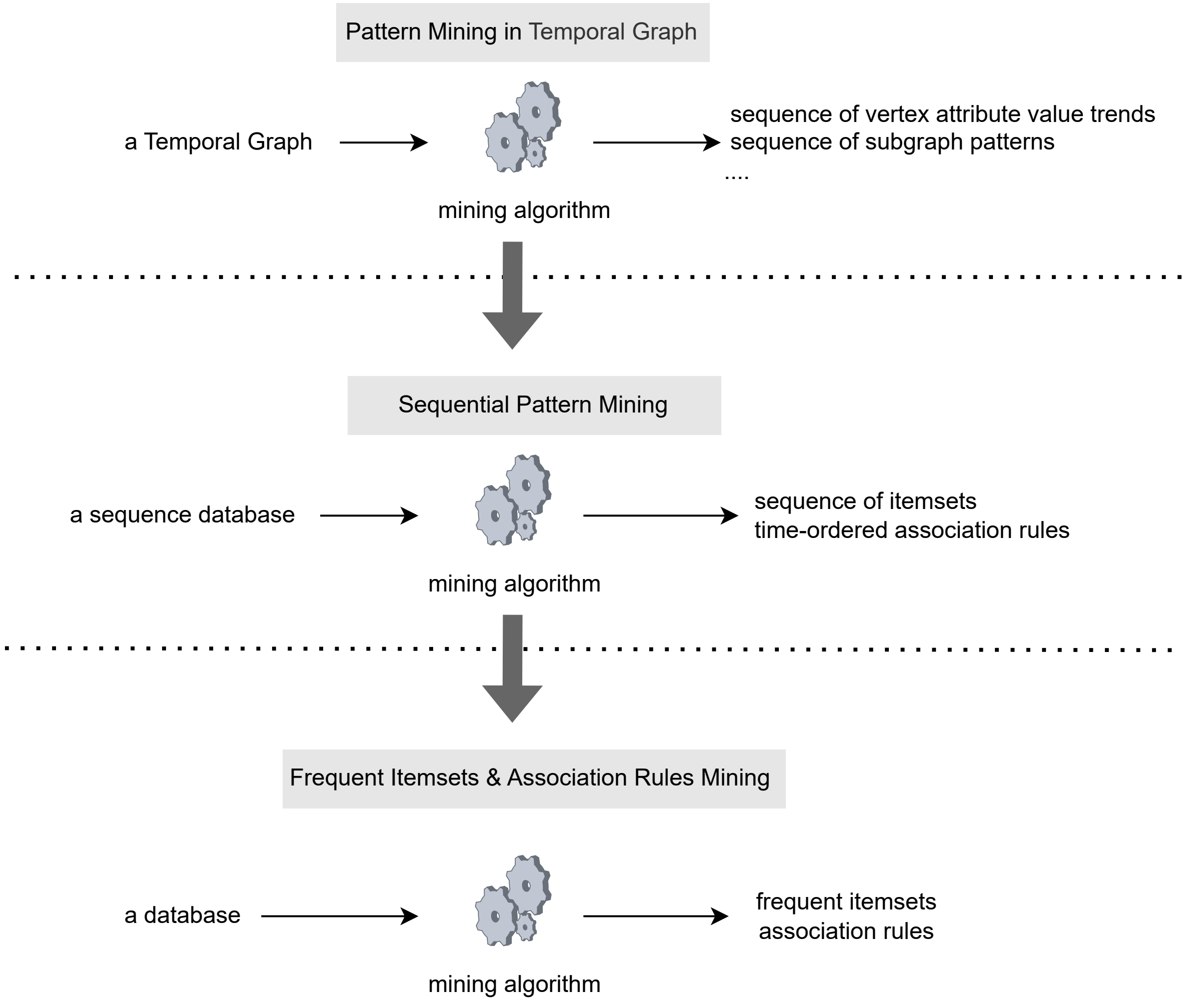

Pattern Mining in Temporal Graph
a Temporal Graph
mining algorithm
sequence of vertex attribute value trends
sequence of subgraph patterns
....
Sequential Pattern Mining
a sequence database
mining algorithm
sequence of itemsets
time-ordered association rules
Frequent Itemsets & Association Rules Mining
a database
mining algorithm
frequent itemsets
association rules

Figure 4.2: Origins of Pattern Mining in Temporal Graphs

Rules Mining and Sequential Pattern Mining to understand the theoretical foundations of Pattern Mining in TG (Section 2.1). Then, we review existing approaches in Pattern Mining in TG (Section 2.2) and make a comparative analysis according to criteria derived from the challenges presented in the previous section (Section 2.3).

2.1 Origins of Pattern Mining in Temporal Graphs

2.1.1 Frequent Itemsets and Association Rules Mining

Pattern Mining is a research field that emerged in the 1990s with the problem of market basket analysis posed by Agrawal et al. (1993, 1994). This problem consists in analysing a *transaction database* about customer transactions to discover *frequent itemsets* (Luna et al., 2019) and *association rules* (Telikani et al., 2020) that help to understand the behaviour of customers. As shown in Table 4.1, a transaction database is a relational table containing a set of records (here, transactions of customers having identifiers TID $= 1, 2, ..., n$) described using a set of binary attributes (here, purchased items). Frequent itemsets are the groups of items appearing together (i.e., bought together) in the database with a support (i.e., the number of transactions containing an itemset) above a given threshold: the minimum support (Luna et al., 2019). For instance, in Table 4.1,

Table 4.1: Transaction database about customer transactions. The column **TID** refers to the transaction identifier.

	Binary Attributes						
TID	milk	bread	butter	beer	diapers	eggs	fruits
1	1	1	1	1	0	0	1
2	0	0	0	1	0	1	1
3	1	1	1	0	1	0	0
4	1	1	1	0	0	1	1
5	0	1	0	0	0	0	0

$\{bread, butter\}$ is a frequent itemset, for a minimal support of 2, as it appears in 3 transactions (transactions 1, 3 and 4). Association rules are strong associations between items in the form of a rule $X \implies Y$ (Telikani et al., 2020). It indicates that some items X are strongly correlated with some other items Y according to constraints (e.g., support and confidence) to define their correlation. For instance, an association rule is $\{bread, butter\} \implies milk$, meaning that every time customers buy butter and bread, they also buy milk.

The *Apriori algorithm* was the first mining algorithm proposed to extract frequent itemsets and association rules from transaction databases (Agrawal et al., 1993, 1994). It takes as input a database similar to Table 4.1 and applies an iterative technique. First, it scans the database to generate candidate itemsets of size 1 (i.e, an itemset composed by each item in the database) and counts their support in the database. If their support is below the minimum support, they are eliminated, as well as their subsets. Otherwise, they are kept in memory to be iterated in the next generation step. Second, it generates candidate itemsets of size k+1 by extending frequent itemsets of size k kept in the previous iteration. It is the *generation* step. Then, it counts the support of candidate itemsets in the database and prunes the infrequent one. It is the *test* step. This *generation-and-test* process is repeated until no further successful extensions are found. The Apriori algorithm uses *Breadth-First Search* (BFS) (Cormen et al., 2022) to generate all $k-$itemsets in each $k-th$ iteration while exploring the search space. However, the Apriori algorithm has three main limitations. First, it requires multiple scans of the database to check the support count of each itemset. So it costs a significant amount of computing power, especially if the database is huge. Second, non-existent candidate itemsets could be generated because candidates are generated just by combining smaller frequent itemsets without accessing the database. Third, due to BFS, a large amount of memory is required to keep all frequent itemsets of size k in memory to generate sequences of length $k + 1$. Over the years, multiple algorithms have been proposed to improve the Apriori algorithm's efficiency (Luna et al., 2019).

Discussion Frequent itemsets and association rule mining approaches ignore the time or sequential ordering of events (Gan et al., 2019). However, it is relevant to consider the ordering of events or elements for providing insights about how an event may be related to another event in a next timestamp. For example, in the market basket analysis, considering the order of transactions allows analysing the evolution of customers' shopping behaviour. To address this issue, the concept of *Sequential Pattern Mining* has emerged.

Table 4.2: Sequence database about customer transactions. The column **SID** refers to the sequence identifier.

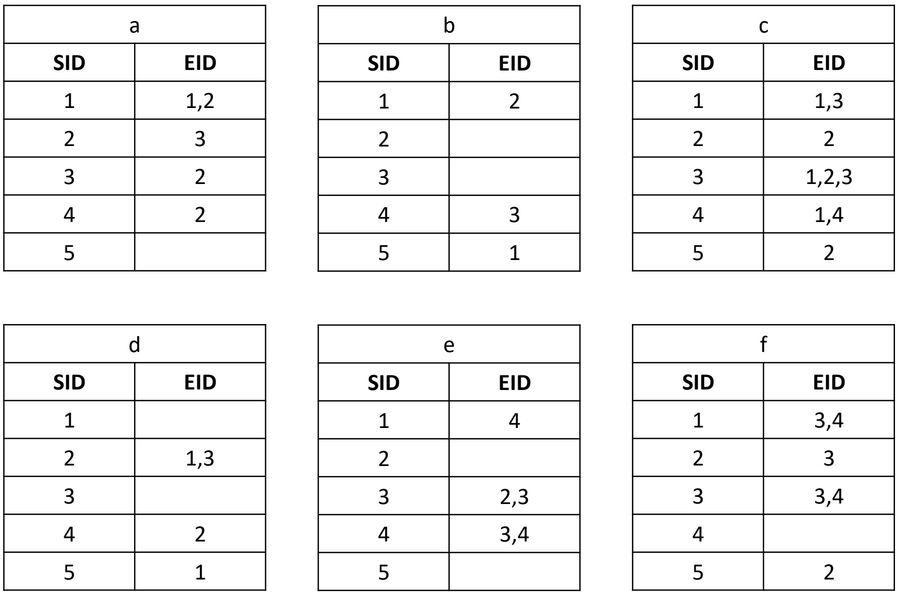
SID	**Sequence**
1	$\langle\{milk, soup\}, \{fruits, cereal, bread\}, \{vegetables, yogurts\}\rangle$
2	$\langle\{diapers\}, \{cotton, baby\,shower\,gel, baby\,milk\,powder\}, \{milk\}, \{cereal, bread\}\rangle$
3	$\langle\{beer, chips\}, \{soda\}, \{pizza, ice\,cream\}\rangle$
4	$\langle\{diapers, milk\}, \{cereal, bread, baby\,milk\,powder\}, \{vegetable\}\rangle$
5	$\langle\{tea, fruits\}, \{honey, vegetables\}, \{soup, bread\}\rangle$

2.1.2 Sequential Pattern Mining

Sequential Pattern Mining was first introduced by Agrawal and Srikant (1995) to find subsequences that appear in a *sequence database* (Table 4.2) with a support (here, the number of sequences containing a subsequence) above a given minimal support. Those subsequences are called *frequent sequential patterns*. A sequence database SDB is a list of sequences $SDB = \langle s_1, s_2, ..., s_p\rangle$ having sequence identifiers (SIDs) $1, 2, ..., p$. A sequence is a time-ordered list of itemsets $s = \langle i_1, i_2, ..., i_n\rangle$ such that $i_k \subseteq I\,(1 \leq k \leq n)$ (Fournier-Viger et al., 2017). For instance, in the sequence database illustrated in Table 4.2, the sequence $\langle milk, \{cereal, bread\}\rangle$ (meaning that customers buy milk and the next time, cereal and bread) is a frequent sequence, for a minimal support of 2, as it appears in 3 sequences (sequences 1, 2 and 4).

All Sequential Pattern Mining algorithms take as input a sequence database and a minimum support threshold (chosen by the user), and as output the set of frequent sequential patterns. The difference between the various algorithms is not their output, but their performance due to the chosen search strategies, data structures and optimizations (Fournier-Viger et al., 2017). There are two main categories of Sequential Pattern Mining approaches: *Apriori* and *Pattern-Growth* approaches. Most sequential Pattern Mining algorithms extend these two main approaches.

Apriori approaches gather algorithms that use the generate-and-test strategy (where candidate patterns are generated, tested and then combined) as in the Apriori algorithm presented in the previous section. Some of the most popular Apriori-based algorithms are *Generalized Sequential Pattern* (GSP) (Srikant and Agrawal, 1996) and *SPADE* (Zaki, 2001). The GSP algorithm scans the database to generate frequent sequences of 1 and keep them in memory. Then it uses the sequences of size k to generate sequences of size $k + 1$. This process is iterated until no sequences could be generated. However, the GSP algorithm has the same limitations as the Apriori algorithm: (i) multiple database scans to calculate the support of each candidate sequence, (ii) the possible generation of non-existent candidate sequences, and (iii) due to the use of BFS, a large amount of memory used to keep all frequent sequences of size k to generate sequences of length $k + 1$ (Fournier-Viger et al., 2017). The SPADE algorithm is an alternative algorithm to avoid the drawbacks of GSP. It can use *Depth-First Search* (DFS) (Cormen et al., 2022) to consume less memory than BFS. Moreover, it uses a vertical representation of a sequence database, as shown in Fig 4.3, to reduce significantly the number of scans of the database. Each table of a vertical database represents the occurrences of an item (a, b, c, d, e or f) in each sequence (SID) with their position in the sequence (EID). The SPADE algorithm generates each sequence of size $k + 1$ by extending a sequence of size k using



a	
SID	EID
1	1,2
2	3
3	2
4	2
5	

b	
SID	EID
1	2
2	
3	
4	3
5	1

c	
SID	EID
1	1,3
2	2
3	1,2,3
4	1,4
5	2

d	
SID	EID
1	
2	1,3
3	
4	2
5	1

e	
SID	EID
1	4
2	
3	2,3
4	3,4
5	

f	
SID	EID
1	3,4
2	3
3	3,4
4	
5	2

Figure 4.3: A vertical representation of a sequence database. The column **SID** refers to the sequence identifier. The column **EID** refers to the position of the item in the sequence SID.

join operations between tables without scanning the database. Similarly, the support of sequences is directly calculated without scanning database. However, SPADE is not efficient to mine a sequence database containing long sequences. The join operations in that case are very costly.

Pattern-Growth approaches can achieve a significant improvement over Apriori-based algorithms. Candidate patterns that do not exist in the database are generated in Apriori-based algorithms. They are generated by combining smaller patterns without accessing the database. In response to this limitation, Pattern-Growth algorithms only consider patterns appearing in the sequence database. To do so, they use the concepts of database projections and DFS. The advantage of database projections is to avoid scanning the whole database when finding the patterns. The most popular Pattern-Growth algorithm is *PrefixSpan* (Jian Pei et al., 2004). As illustrated in Fig 4.4, first, it scans the sequence database to generate all frequent sequences of size 1. Second, it projects the database on the prefix (i.e, frequent patterns found earlier) to create a projected database containing the prefix. Third, it extends the prefix with the items which are frequent in the projected database to form longer sequential patterns. This process is then recursively repeated as a DFS to find all frequent sequential patterns. Although Pattern-Growth algorithms can achieve a significant improvement over Apriori-based algorithms, the cost of scanning databases and performing projections can be quite high in terms of memory and time.

To optimize sequential mining algorithms' performances, researchers have also proposed to integrate constraints in the mining process. Constraints enable to reduce the search space during the extraction of patterns. Constraints are additional criteria on the patterns to be found. A classic constraint is the frequency or minimum support of patterns we discussed earlier. The GSP algorithm (Srikant and Agrawal, 1996) introduces the constraints of minimum and maximum amount of time windows between two consecutive itemsets (gap constraints), and a maximum time duration for each sequential pattern (du-

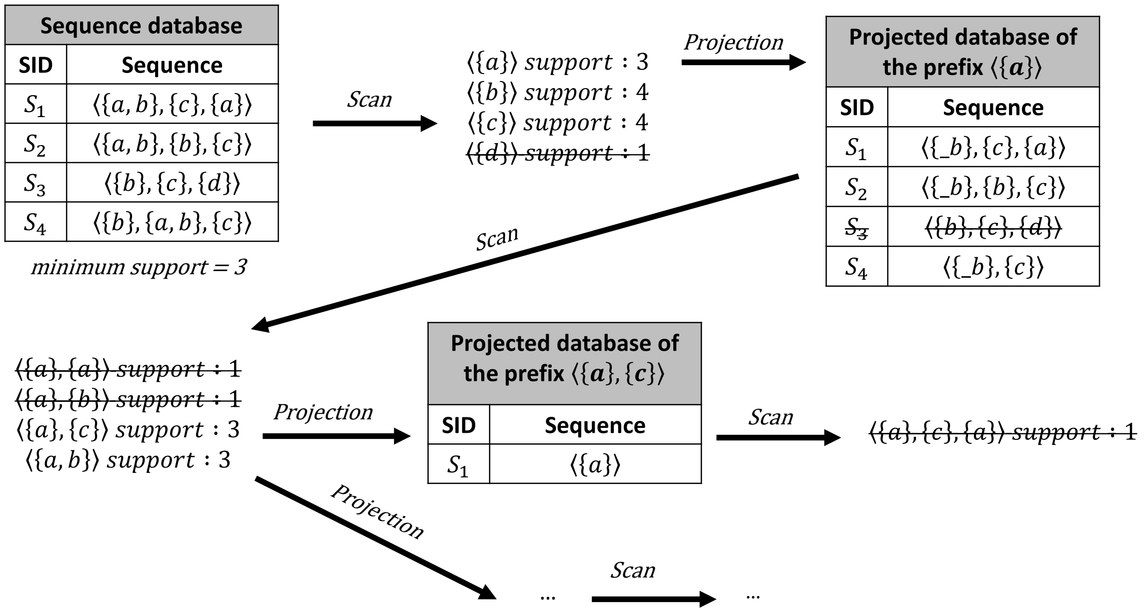

Sequence database
SID | Sequence
S_1 | $\langle\{a,b\},\{c\},\{a\}\rangle$
S_2 | $\langle\{a,b\},\{b\},\{c\}\rangle$
S_3 | $\langle\{b\},\{c\},\{d\}\rangle$
S_4 | $\langle\{b\},\{a,b\},\{c\}\rangle$
minimum support = 3
Scan
$\langle\{a\}\rangle$ support : 3
$\langle\{b\}\rangle$ support : 4
$\langle\{c\}\rangle$ support : 4
~~$\langle\{d\}\rangle$ support : 1~~
Projection
Projected database of the prefix $\langle\{a\}\rangle$
SID | Sequence
S_1 | $\langle\{\_b\},\{c\},\{a\}\rangle$
S_2 | $\langle\{\_b\},\{b\},\{c\}\rangle$
~~S_3~~ | ~~$\langle\{b\},\{c\},\{d\}\rangle$~~
S_4 | $\langle\{\_b\},\{c\}\rangle$
Scan
~~$\langle\{a\},\{a\}\rangle$ support : 1~~
~~$\langle\{a\},\{b\}\rangle$ support : 1~~
$\langle\{a\},\{c\}\rangle$ support : 3
$\langle\{a,b\}\rangle$ support : 3
Projection
Projected database of the prefix $\langle\{a\},\{c\}\rangle$
SID | Sequence
S_1 | $\langle\{a\}\rangle$
Scan
~~$\langle\{a\},\{c\},\{a\}\rangle$ support : 1~~
Projection
...
Scan
...

Figure 4.4: Process of PrefixSpan algorithm

ration constraint). Pei et al. (2007) also propose various constraints in Pattern-Growth algorithms, such as items that should appear or not in sequential patterns (item constraints) and minimum and maximum number of items per sequential patterns (length constraints).

2.2 Pattern Mining in Temporal Graphs

In the 2000s, the research field of Sequential Pattern Mining has evolved to consider the temporality of a new type of data: Temporal Graphs (TG). Indeed, TG open the opportunity of discovering novel patterns: evolution mechanisms capturing information from the multiple dimensions of TG (topology, attributes, time). In the field of Pattern Mining in TG, various types of TG are defined based on the information in patterns researchers aim to capture. First, we have the '*dynamic graphs*'. They designate a sequence of time-ordered graphs allowing to represent the evolution of a graph over a set of timestamps $\{t_1, t_2, ..., t_{max}\}$ (Fournier-Viger et al., 2020a). Vertices and edges of each graph of the sequence represent respectively entities and relationships (or other interactions) between entities that are valid at a timestamp. Extracted patterns from dynamic graphs capture essentially information from the topology evolution of TG. To enrich the extracted patterns, '*dynamic labelled graphs*' extend dynamic graphs by associating a label on each vertex or edge to describe an attribute of an entity (or relationship) (Fournier-Viger et al., 2020a). Extracted patterns from dynamic labelled graphs may thus capture information from the evolution of topology and one attribute of vertices (or edges). Finally, the richest representation of TG is '*dynamic attributed graphs*'. Dynamic attributed graphs represent also an extension of dynamic graphs (Fig 4.5). Several attributes can be associated to vertices instead of one label to provide more semantics to vertices. Patterns extracted in those dynamic attributed graphs may therefore capture the evolution of topology and several vertex attributes. **It is important to notice that dynamic attributed graphs represent the most complete type of TG in terms of evolution. They are there-**

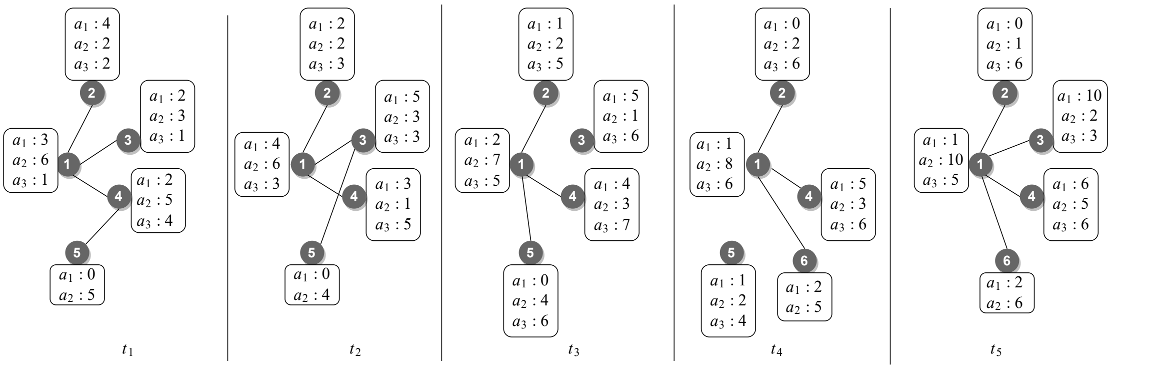


Figure 4.5: A dynamic attributed graph having four timestamps with numerical attribute values

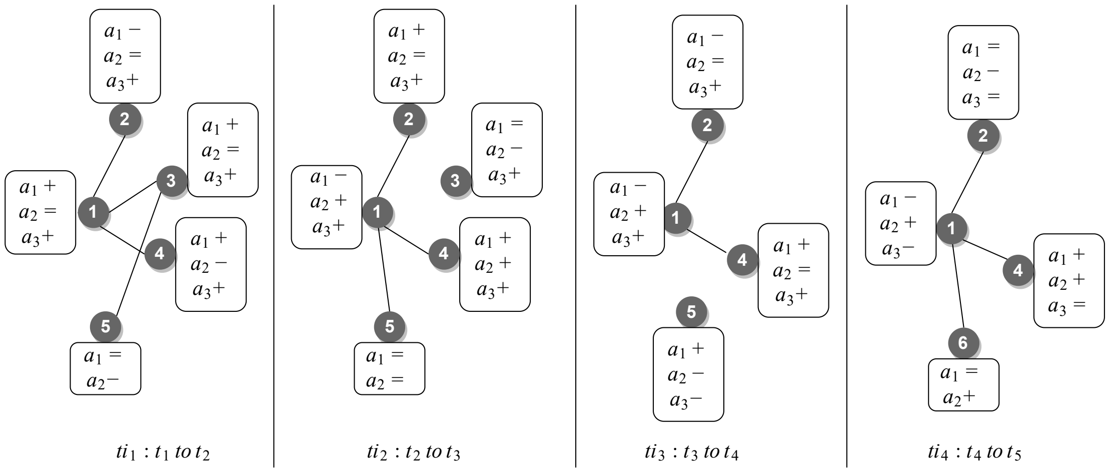


Figure 4.6: A sequence of trend graphs

fore the closest to our TG model in Chapter 2. Consequently, in the following, we will focus on Pattern Mining techniques within dynamic attributed graphs as a foundational step towards adapting and expanding these techniques to suit our TG model.

Mining patterns in dynamic attributed graphs consists of finding complex relationships between topology evolution (i.e., addition/removal of edges over time) and vertex attribute evolution (i.e., addition/removal and update of attributes over time) to highlight some evolution mechanisms. To take into account attribute value evolution in patterns, a common practice is to convert numerical attribute values into trends (increase, decrease or equality) to show how they have changed over time. For instance, Fig 4.6 shows the result of converting the dynamic attributed graph in Fig 4.5 into a sequence of trend graphs. To the best of our knowledge, all studies on Pattern Mining in dynamic attributed graphs use the sequence of trend graphs as input Fournier-Viger et al. (2020a). In the following, we analyse the existing approaches of Pattern Mining in dynamic attributed graphs.

2.2.1 Cohesive Co-evolution Patterns

The problem of mining cohesive co-evolution patterns in dynamic attributed graphs is proposed by Desmier et al. (2012). A *cohesive co-evolution pattern* is a set of vertices that

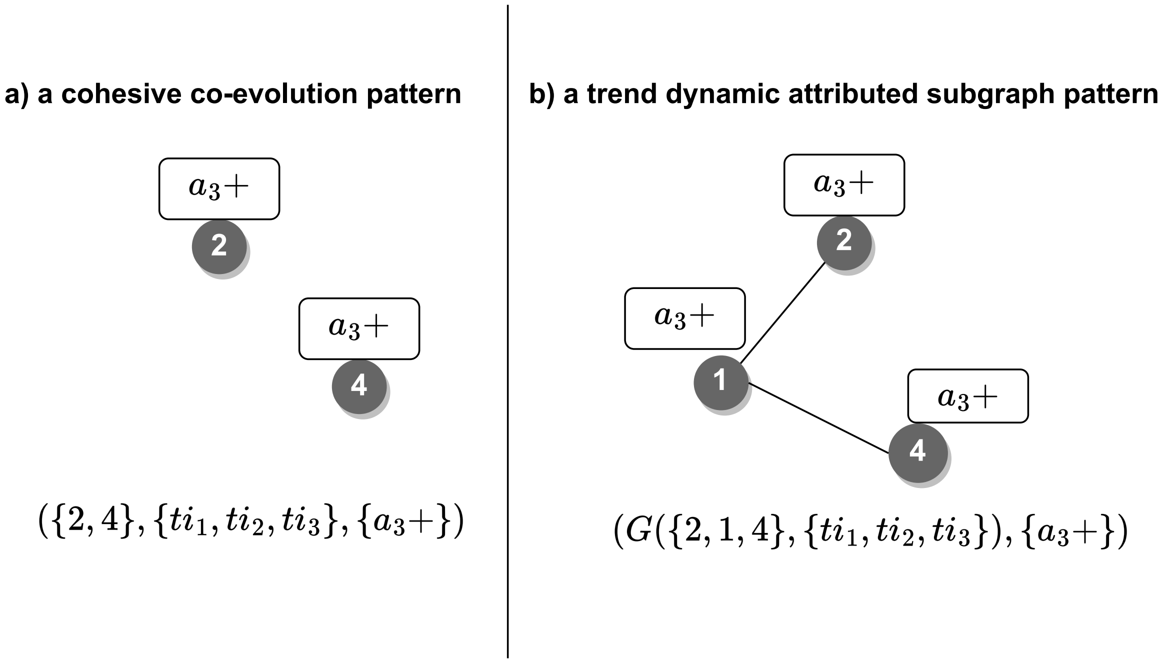

a) a cohesive co-evolution pattern
a_3+
2
a_3+
4
$(\{2,4\},\{ti_1,ti_2,ti_3\},\{a_3+\})$
b) a trend dynamic attributed subgraph pattern
a_3+
2
a_3+
1
a_3+
4
$(G(\{2,1,4\},\{ti_1,ti_2,ti_3\}),\{a_3+\})$

Figure 4.7: A cohesive co-evolution pattern (a), a trend dynamic attributed subgraph (b)

have similar neighbourhood and similar trend values (i.e., increase, decrease or equality) for some attributes over a set of timestamps. It is a triplet (V, T, Ω) where V is a set of vertices, T is a set of timestamps and Ω is a set of signed attributes (i.e., trends attached to attributes). For instance, $(\{2, 4\}, \{ti_1, ti_2, ti_3\}, \{a_3+\})$ in Fig. 4.7 a) is a cohesive co-evolution pattern extracted from the dynamic attributed graph of Fig 4.6. This pattern indicates that the attribute value of a_3 of vertices 2 and 4 have seen their value increased over ti_1, ti_2 and ti_3. In addition, several constraints have been proposed by the authors to reduce space search. Specifically, the cohesiveness constraint imposes a minimum similarity of the direct neighbourhood of pairs of vertices, based on similarity measures such as Cosine similarity. Finally, they propose an algorithm to extract cohesive co-evolution patterns. All patterns are enumerated in a DFS manner by recursively appending vertices to patterns and checking that constraints are satisfied.

2.2.2 Trend Dynamic Attributed Subgraph Patterns

Desmier et al. (2013) extend cohesive co-evolution patterns in Desmier et al. (2012). They propose the new pattern of *trend dynamic attributed subgraph* denoted as, $(G(V, T), \Omega)$ where $G(V, T)$ is a dynamic subgraph occurring at each timestamp of T and Ω is a set of signed attributes. In other words, a trend dynamic attributed subgraph is a set of connected vertices respecting some constraints on topology and trends of attribute values. For instance, $(G(\{2, 1, 4\}, \{ti_1, ti_2, ti_3\}), \{a_3+\})$ in Fig. 4.7 b) is a trend dynamic subgraph extracted from the dynamic attributed graph of Fig 4.6. As topology is underused in Desmier et al. (2012) (exploited through a similarity measure), the authors integrate a new constraint on topology Desmier et al. (2013). The connectivity of the dynamic subgraphs is constrained by a maximum diameter value that limits the length of the path between two vertices of a dynamic attributed subgraph. The authors develop the *MINTAG* algorithm based on a DFS exploration strategy to extract the pattern.

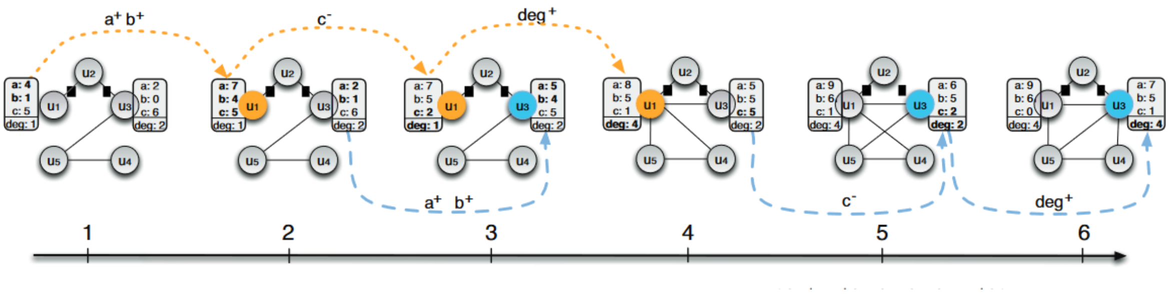


Figure 4.8: A triggering pattern $\langle\{a+, b+\}, \{c-\}, \{deg+\}\rangle$ supported by the vertices in yellow ($u1$) and blue ($u3$). Source: Kaytoue et al. (2014)

2.2.3 Triggering Patterns

Kaytoue et al. (2014) define the triggering pattern which is a sequence $P = \langle L, R\rangle$ where L is a sequence of attribute variations of a vertex followed by a single variation of a topological property (degree, betweenness, number of clicks etc.) of the same vertex denoted as R. In Fig 4.8 the triggering pattern $\langle\{a+, b+\}, \{c-\}, \{deg+\}\rangle$ indicates that an increase in the attribute value of a and b followed by a decrease in the attribute value of c in the next time, triggers an increase of the degree property (number of incident edges to a vertex). This pattern is supported by two vertices, $u1$ and $u3$. The authors define two constraints on such patterns: the support and the growth rate. The support constraint is defined by the number of vertices that satisfies the pattern, while the growth rate gives the discriminating power of the sequence variations to explain a topological change. They design the algorithm *TRIGAT* to mine triggering patterns. First, *TRIGAT* generates all 1-item sequences by checking the previous constraints. Then, it extends patterns using a Pattern-Growth approach (Section 2.1.2).

2.2.4 Significant Trend Sequences

Fournier-Viger et al. (2019) address a limitation of triggering patterns (Kaytoue et al., 2014). Indeed, the calculation of the influence of the attribute variations on the topological change in a triggering pattern is only based on the last attribute variation. Thus, this approach may find patterns where attribute variations are weakly correlated with each other over time. Therefore, they propose a new pattern named *significant trend sequence*. They consider the very specific case where a vertex's attributes influence its neighbours' attributes. A significance measure, called Sequence Virtual Growth Rate, is proposed with the pattern to measure the correlation between all consecutive attribute variations of a sequence. For example, in Fig 4.9, $\langle\{a1+, a2+\}, \{a3-\}\rangle$ is a significant trend sequence. $\{a3-\}$ is strongly correlated with $\{a1+, a2+\}$ because $\{a1+, a2+\}$ is very often followed by $\{a3-\}$ but not because $\{a3-\}$ is globally frequent. To mine significant trend sequences, two algorithms, named $TSeqMiner_{dfs-dfs}$ and $TSeqMiner_{dfs-bfs}$, are proposed. Both algorithms are based on a Pattern-Growth approach.

2.2.5 Attribute Evolution Rules

Fournier-Viger et al. (2020b) propose a pattern named *attribute evolution rule*. An attribute evolution rule (AER) is a tuple $R : (V, E, \lambda_{before}, \lambda_{after})$ that indicates how the attribute values (λ) of a connected subgraph (V, E) have evolved for two consecutive timestamps. For example, $(\{x, y, z\}, \{(x, y), (y, z)\}, \{x : a+, z : b-\}, \{y : c+, d-\})$ in

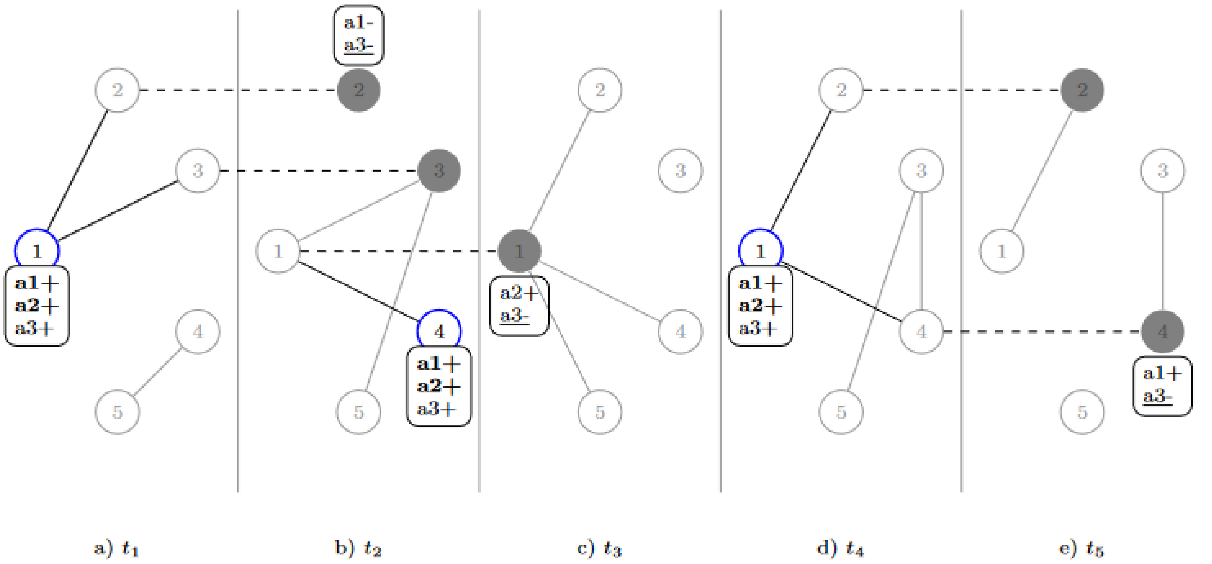


Figure 4.9: A significant trend sequence $\langle\{a1+, a2+\}, \{a3-\}\rangle$. Source: Fournier-Viger et al. (2019)

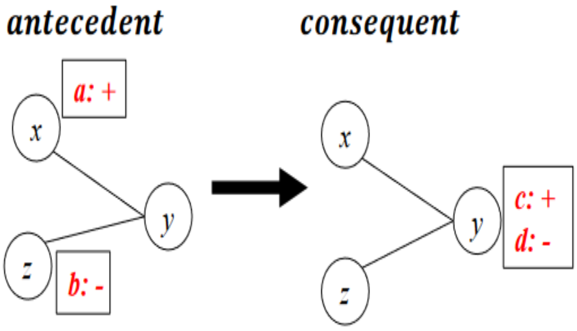


Figure 4.10: An attribute evolution rule $(\{x, y, z\}, \{(x, y), (y, z)\}, \{x : a+, z : b-\}, \{y : c+, d-\})$. Source: Fournier-Viger et al. (2020b)

Fig 4.10 is an attribute evolution rule. It indicates that the attribute a of vertex x has increased $(a+)$ and the attribute b of vertex z has decreased $(b-)$, which then caused vertex y's attribute c to increase and attribute d to decrease at the next timestamp. The algorithm $AER - miner$ is proposed to extract AERs from a dynamic attributed graph. It is based on an Apriori-based approach. First, it generates core patterns (a consequent vertex with an attribute). Then, core patterns are extended iteratively (adding an antecedent vertex with an attribute) using the generate-and-test strategy and BFS. Then, core patterns are merged to obtain the AERs.

2.2.6 Recurrent Patterns

Cheng et al. (2017) generalize the concept of pattern in a dynamic attributed graph by proposing recurrent patterns. A *recurrent pattern*, denoted as $P = (S_P, T_P)$, is a sequence of subgraphs S_P which represents recurring evolutions of connected vertices regarding their attribute values over a set of timestamps T_P. Each subgraph can be described using different vertex attributes and trends. For instance, in Fig. 4.11, $(\langle(1 : a_3 + |2 : a_2 =$

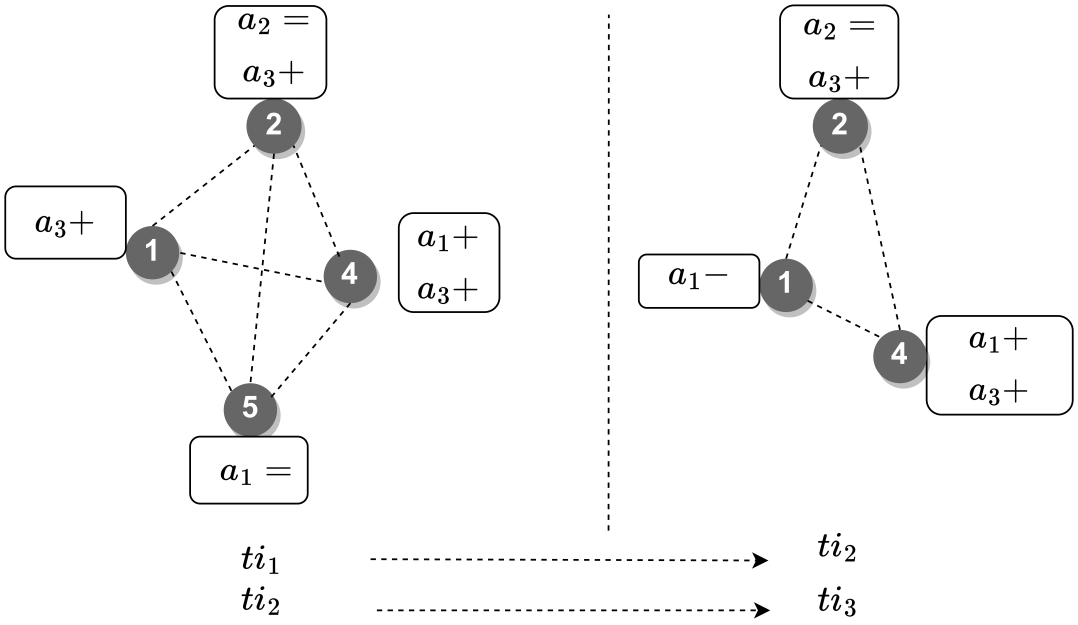


Figure 4.11: A recurrent pattern

$a_3 + |4 : a_1 + a_3|5 : a_1 =)(1 : a_1 - |2 : a_2 = a_3 + |4 : a_1 + a_3)\rangle, \{ti_1, ti_2\})$ is an example of a recurrent sequence of subgraphs starting at times ti_1 and ti_2 extracted from the dynamic attributed graph of Fig 4.6. In other words, the sequence appears at the sequence of times ti_1, ti_2 and at ti_2, ti_3.

The authors consider several constraints such that a pattern is relevant. A pattern must be connected vertices (connectivity), appear a minimum number of times (frequency), have a minimum number of vertices (volume) and describe a common core of individuals to follow evolution (temporal continuity). Then, they proposed the *RPMiner* algorithm. It is based on an iterative and incremental approach. First, it iteratively constructs size-1 patterns. To do so, it performs different combinations of timestamps, traverses the dynamic attributed graph to find subgraphs satisfying frequency constraints and volume constraints and finally performs intersections of these subgraphs at these timestamp combinations. Second, for each time combination, it tries to extend incrementally each pattern generated from the previous iteration with the current size-1 patterns by checking the temporal continuity constraint.

2.3 Comparative Analysis of Pattern Mining Approaches in Temporal Graphs

Considering the challenges we presented in Section 1.2, we want to extract patterns in dynamic attributed graphs that are (i) representative (i.e., capturing evolution across several groups of vertices), (ii) sequential (i.e, describing a sequence of events to capture temporal relationships between events), (iii) complete (i.e., fully capturing the information from evolution) and (iv) using an efficient algorithm. Therefore, we compare Pattern Mining approaches in dynamic attributed graphs according to the characteristics of the pattern and the strategy used in the mining algorithm. More precisely, we define the following comparison criteria of patterns:

- The *subject* described in the pattern evaluates if the described evolution mechanism concerns a single vertex, a group of vertices or several groups of vertices;
- The *time span* of the pattern evaluates if the described evolution mechanism is within a single time point describing an event or within a time interval (or a sequence of time points) describing a sequence of events ordered over time;
- the *evolution aspects* included in the pattern evaluates if the described evolution mechanism captures *fully, partially or not at all* the information from the evolution of vertex attributes and topology;

Table 4.3 shows the comparison of these approaches according to the previous criteria.

2.3.1 Patterns

Cohesive co-evolution patterns Desmier et al. (2012) and *Trend dynamic attributed subgraphs patterns* (Desmier et al., 2013) describe the evolution mechanism of a group of vertices within a single time point. Therefore, these patterns are neither representative nor sequential. Besides, since they are not sequential, they consider vertices having the same trend in their attribute values over time. Thus, they are limited in capturing the evolution of vertex attributes, since vertices can follow different value trends over time. Moreover, they integrate some constraints on topology but do not capture topology evolution.

The mining problem of *triggering patterns* in Kaytoue et al. (2014) allows finding temporal relationships between the changes in attribute values and the change of a topological property of a single vertex. Triggering patterns are therefore sequential but not representative. They focus indeed on the evolution of a single vertex. Regarding the evolution aspects, triggering patterns enable to follow the different trends of attribute values over time. However, the evolution of topology is only represented through the change in the topological properties of vertices (e.g., degree).

Fournier-Viger et al. (2019) propose the mining problem of *significant trend sequences* to discover the influence of the changes in attribute values of a single vertex on its neighbours in the next timestamp. In other words, significant trend sequences describe a sequential evolution within a group of vertices. So they are not representative. Regarding evolution aspects, they capture the changes in attributes but do not include any aspects on the evolution of topology.

Fournier-Viger et al. (2020b) define the mining problem of *attribute evolution rules* to discover the influence of attribute changes within a group of vertices over time. Attribute evolution rules are sequential, but not representative. Regarding evolution aspects, they capture the changes in attribute values but consider the same topology (i.e, same sets of vertices and edges) over time. For instance, for an epidemic within a population, significant trend sequences represent the evolution of the same group of persons without any changes in the composition of the group in terms of persons or interactions. It is impossible to know if new interactions between persons (i.e., edges) or new persons (i.e., vertices) contribute to the propagation of the epidemic. So they ignore the evolution of topology.

Recurrent patterns proposed by Cheng et al. (2017) describe the sequential evolution

Table 4.3: Comparison of different pattern mining problems

	Pattern Type							Mining Strategy
	Evolution Subject			Time Span		Evolution Aspects		
	A single vertex	A group of vertices	Several groups of vertices	Single time point	Time interval (sequence)	Vertex Attributes	Topology	
Cohesive Co-evolution Patterns Desmier et al. (2012)		●		●		◐		DFS
Trend Dynamic Attributed Subgraph Patterns Desmier et al. (2013)		●		●		◐		DFS
Triggering Patterns Kaytoue et al. (2014)	●				●	●	◐	pattern-growth
Recurrent Patterns Cheng et al. (2017)		●			●	●	●	intersections and extensions
Significant Trend Sequences Fournier-Viger et al. (2019)		◐			●	●		pattern-growth
Attribute Evolution Rules Fournier-Viger et al. (2020b)		●			●	●		A priori-based

of the attribute values of a group of connected vertices or 'subgraphs'. Contrary to the previous work, they integrate the evolution of topology by allowing a different structure of subgraphs of the sequence. More precisely, two consecutive subgraphs of a sequence can have new vertices and edges as long as they have a common core (temporal continuity constraint). However, even if they fully capture the evolution of attribute values and topology, they focus on the evolution of a group of connected vertices. So they are not representative. The reason is that recurrent patterns represent the attribute values and topology evolution of a specific group of vertices that is temporally frequent. They do not look at the spatial frequency of the evolution, i.e., if the evolution is spatially located in other groups of vertices.

2.3.2 Mining Strategy

What all the previous work has in common is that mining algorithms require making repetitive traversals of the search space to generate patterns. These repetitive traversals increase the computational complexity (time and memory) of these algorithms significantly as the search space grows (i.e, the pattern size or graph size grows). Therefore, researchers propose different mining strategies and constraints, such as the support (or frequency) of patterns, to reduce the need for traversing the search space. They mostly apply strategies found in frequent itemset mining (Section 2.1): DFS (Desmier et al., 2012, 2013), Pattern-Growth approach (Kaytoue et al., 2014; Fournier-Viger et al., 2019) and Apriori-based approach (Fournier-Viger et al., 2020b). These algorithms treat attribute values of vertices as itemsets to generate the patterns. They only consider topology as constraints or as attributes. The algorithm for extracting recurrent patterns (Cheng et al., 2017) is more complex than previous ones. The algorithm fully exploits the evolution of topology in addition to the evolution of attribute values. Thus, the algorithm explores attribute values of vertices and traverse the topology to generate subgraphs instead of itemsets.

2.3.3 Conclusion

After analysing existing approaches to mine patterns in dynamic attributed graphs, we first observe in Table 4.3 that no pattern is representative, i.e, represents the evolution of several groups of vertices. Second, most patterns are sequential to highlight the temporal relationships between events. Third, except for recurrent patterns Cheng et al. (2017), current patterns are incomplete in terms of the evolution aspects they capture. **It is therefore necessary to define a novel pattern which is representative, sequential and complete.**

The previously mentioned algorithms do not allow extracting this novel pattern, especially because they count the frequency of patterns only temporally. Mining such patterns in a dynamic attributed graph is therefore potentially more complex than mining recurrent patterns. **It is therefore necessary to propose a mining strategy that can both guarantee the mining efficiency and the representativeness, completeness and sequentiality of patterns.**

3 Frequent Sequential Subgraph Evolutions (FSSE) and Problem Setting

As mentioned in the previous sections, a new Pattern Mining problem needs to be defined to meet the challenges we posed. In this section, we present our Pattern Mining problem.

We first give a definition of the *dynamic attributed graph* (Section 3.1). Second, we define a novel pattern, called *frequent sequential subgraph evolutions (FSSE)* (Section 3.2). Third, we define several constraints helping to reduce the space search (Section 3.3). Finally, we formally define the problem setting (Section 3.4).

3.1 Dynamic Attributed Graph

The input dataset of our Pattern Mining problem is a dynamic attributed graph. In this section, we give a precise definition of a dynamic attributed graph and also detail a necessary pre-processing phase of the dynamic attributed graph.

Definition 22 (dynamic attributed graph)**.** *A dynamic attributed graph, denoted as* $\mathcal{G} = \langle G_{t_1}, G_{t_2}, ..., G_{t_{max}} \rangle$*, is a sequence of graphs that represents the evolution of a graph over a set of ordered and consecutive timestamps* $T = \{t_1, t_2, ..., t_{max}\}$*. It is composed of the set of vertices denoted as* $\mathcal{V}$*. The set of attributes* $\mathcal{A}$ *is used to describe all the vertices. Each attribute* $a \in \mathcal{A}$ *is associated with a value domain. A value domain* $\mathbb{D}_a$ *(numerical or categorical) is associated to each vertex and attribute* $a \in \mathcal{A}$*. So* $\mathbb{D} = \cup_{a \in \mathcal{A} \mathbb{D}_a}$*. For each time* $t \in T$*,* $G_t = (V_t, E_t, \lambda_t)$ *is an attributed undirected graph where:* $V_t \subseteq \mathcal{V}$ *is the set of vertices,* $E_t \subseteq V_t \times V_t$ *is the set of edges, and* $\lambda_t : V_t \to 2^{\mathcal{A}\mathbb{D}}$ *is a function that associates each vertex of* V_t *with a set of attribute values* $\mathcal{A}\mathbb{D} = \cup_{a \in \mathcal{A}}(a \times \mathbb{D}_a)$*. The value of the attribute* a *at time* t *of vertex* v *is denoted as* $G_t(v, a)$*.*

Example 8. *Fig. 4.12 presents an example of a dynamic attributed graph with* $\mathcal{V} = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21\}$*,* $T = \{t_0, t_1, t_2, t_3\}$*,* $\mathcal{A} = \{a_1, a_2\}$ *where* a_1 *and* a_2 *are numerical attributes with the domain* $\mathcal{D}_{a_1} = \mathbb{N}$ *and* $\mathcal{D}_{a_2} = \mathbb{N}$*. We have* $G_{t_1}(4, a_1) = 10$*.*

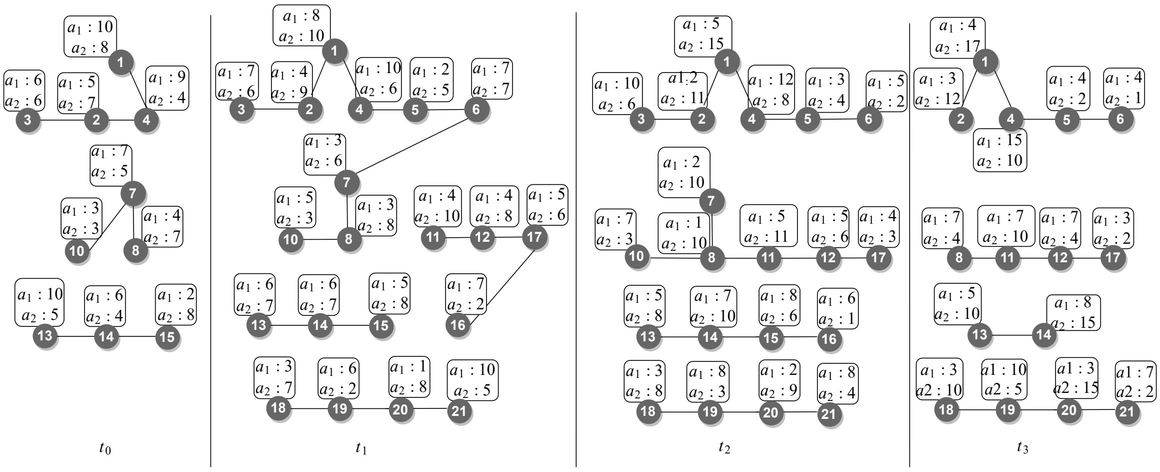


Figure 4.12: Dynamic attributed graph

To study how attribute values of vertices change over time, the concept of trend is defined. When attribute values of vertices are *numerical*, they are usually converted to trend values, since their absolute variations are not interesting (Fournier-Viger et al., 2020a).

Definition 23 (trend)**.** *A trend is an increase* (+)*, decrease* (−) *or stability* (=) *of the value of a vertex's attribute between two consecutive timestamps* t_i *and* t_{i+1}*. A trend set*

Algorithm 4: Pre-processing of the dynamic attributed graph

```
Input: $\mathcal{G}$ : a dynamic attributed graph
Output: $\mathcal{G}'$ : a dynamic attributed graph with trends
Create a new dynamic attributed graph $\mathcal{G}'$
foreach $G_{t_i} \in \mathcal{G}$ do
    $V'_{t_{i+1}} \leftarrow \emptyset$
    $E'_{t_{i+1}} \leftarrow \emptyset$
    foreach $v_j \in G_{t_i} \in \mathcal{G}$ do
        if $\exists v_j \in G_{t_{i+1}} \in \mathcal{G}$ then
            /* Create a new vertex $v'_j$ */
            $v'_j \leftarrow \emptyset$
            foreach $a_k \in v_j$ do
                /* Get the trend of the attribute values of $a_k$ between $t_i$ and $t_{i+1}$ */
                $a'_k \leftarrow ts(a_k, [t_i, t_{i+1}])$
                /* Add the attribute with its trend in the new vertex $v'j$ */
                $v'_j \leftarrow v'_j \cup \{a'_k\}$
            $V'_{t_{i+1}} \leftarrow V'_{t_{i+1}} \cup \{v'_j\}$
    $E'_{t_{i+1}} \leftarrow E_{t_{i+1}}(V'_{t_{i+1}})$
    $G'_{t_{i+1}} \leftarrow (V'_{t_{i+1}}, E'_{t_{i+1}})$
    $\mathcal{G}' \leftarrow \mathcal{G}' \cup G'_{t_{i+1}}$
Return $\mathcal{G}'$
```

of a vertex v is the set of trends of all of its attributes during a time interval $[t_i, t_{i+1}]$. It is denoted as $ts(v, [t_i, t_{i+1}])$.

Example 9. *Consider the vertex 1 during the time interval $[t_0, t_1]$. The trend set of the vertex 1 is $ts(1, [t_0, t_1]) = \{a_1-, a_2+\}$. In other words, the value of a_1 decreases from t_0 to t_1 while the value of a_2 increases.*

Pre-processing the dynamic attributed graph is necessary before the mining process, in the case we have numerical attribute values. The pre-processing phase consists in constructing a new graph $G'_{t_{i+1}}$ for each pair of graphs G_{t_i} and $G_{t_{i+1}} \in \mathcal{G}$. To construct each graph $G'_{t_{i+1}}$, vertices and edges in the graph $G_{t_{i+1}}$ are kept, except for the vertices that only appear in G_{t_i} but not in $G_{t_{i+1}}$ (or the opposite). Indeed, it is not possible to derive trend values from a vertex that appear only in one of the two consecutive timestamps. Then, the values of numerical attributes of the dynamic attributed graph are transformed into trends derived from the difference of each pair G_{t_i}, $G_{t_{i+1}}$. Finally, we end up with $|T| - 1$ timestamps. The algorithm 4 describes this pre-processing phase.

Example 10. *Fig. 4.13 depicts the dynamic attributed graph in Fig. 4.12 after pre-processing. As the values of attributes a_1 and a_2 are numerical in the initial graph, they were transformed to trend values such that $\mathbb{D}_a = \{+, -, =\}$. The pre-processed dynamic attributed graph in Fig. 4.13 is composed of three graphs at t_1, t_2 and t_3. They represent respectively (i) vertices present both in t_0 and t_1, edges of t_1 and trend values of vertex attributes from t_0 to t_1 of the dynamic attributed graph in Fig. 4.12, (ii) vertices present both in t_1 and t_2, edges of t_2 and trend values of vertex attributes from t_1 to t_2 of the dynamic attributed graph in Fig. 4.12, (iii) vertices present both in t_2 and t_3, edges of t_3 and trend values of vertex attributes from t_2 to t_3 of the dynamic attributed graph in Fig. 4.12.*

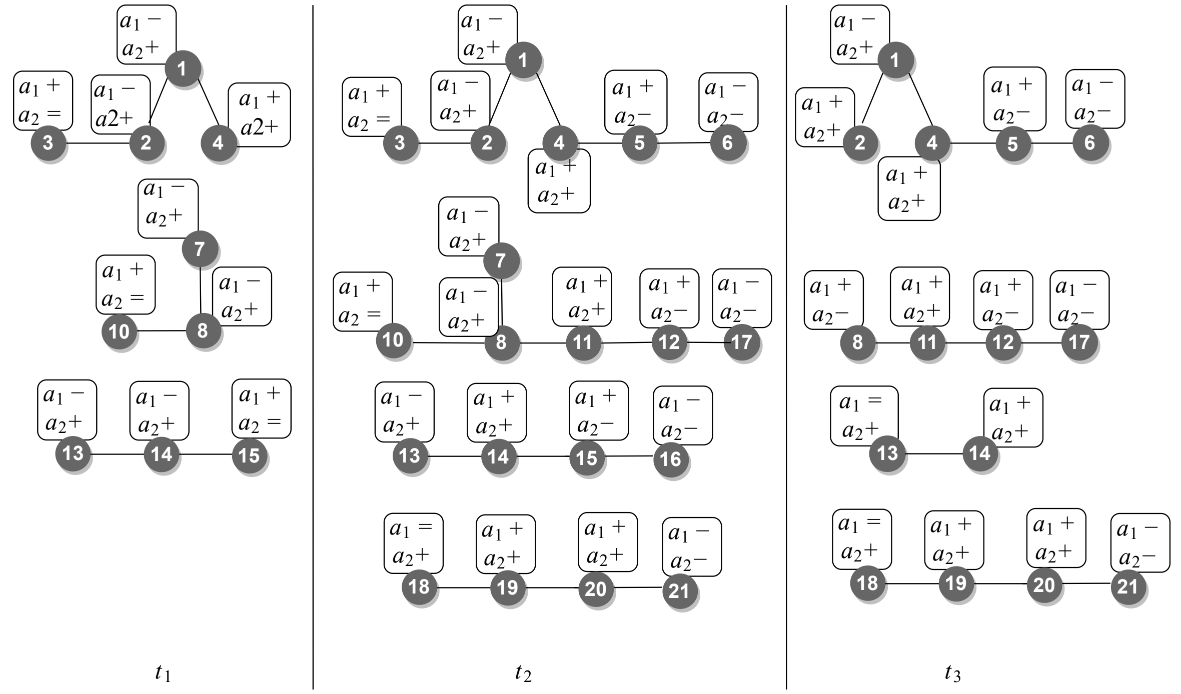


Figure 4.13: Dynamic attributed graph after pre-processing

3.2 A New Pattern

In this section, we propose the novel pattern to be extracted in dynamic attributed graphs: *frequent sequential subgraph evolutions (FSSE)*. The objective for the pattern is to be : (i) representative (i.e. it accounts for the evolution of several groups of vertices), (ii) sequential (i.e. it describes a sequence of events in order to account for the temporal relationships between events) and (iii) complete (i.e. it accounts for all the information derived from the evolution of the graph).

Definition 24 (occurrence)**.** *An occurrence of the set of attribute values $\lambda = (\lambda_1, \lambda_2, ..., \lambda_n)$ (trend or categorical values) in a timestamp $t \in T$ is a group of connected vertices $(v_1, v_2, ..., v_n)$ in $G_t \in \mathcal{G}$ having the set of attribute values λ such that each $\lambda_i \in \lambda$ is the attribute value of the i-th vertex in the set of connected vertices. It is denoted as :*

$$Occurrence(\lambda)\, in\, t = \{t : (v_1, v_2, ..., v_n)\}$$

The occurrences of the set of attribute values $\lambda = (\lambda_1, \lambda_2, ..., \lambda_n)$ in a set of timestamps $T' \subseteq T$ refer to several groups of connected vertices over $\forall G_t, t \in T'$ having the set of attribute values λ. They are therefore denoted as :

$$Occurrence(\lambda)\, in\, T' = \{t_j : (v_1, v_2, ..., v_n)|t_j : (v_1', v_2', ..., v_n')...|t_k : (v_1", v_2", ..., v_n")\}$$

such that $\forall t \in T', Occurrence(\lambda)\, in\, t \subseteq G_t \in \mathcal{G}$. Occurrences are separated by vertical bars and prefixed by their timestamp.

Example 11. *Consider the set of attribute values $\lambda = (a_1 - a_2+, a_1 - a_2+, a_1 + a_2 =)$. The occurrences of λ in t_1 and t_2 in the dynamic attributed graph in Fig. 4.13 are $Occurrence(\lambda)\, in\, T' = \{t_1 : (1, 2, 3)|t_1 : (7, 8, 10)|t_1 : (13, 14, 15)|t_2 : (1, 2, 3)|t_2 : (7, 8, 10)\}$ where $T' = \{t_1, t_2\}$. More precisely, we have $Occurrence(\lambda)\, in\, t_1 = \{t_1 : (1, 2, 3)|t_1 : (7, 8, 10)|t_1 : (13, 14, 15)\}$ and $Occurrence(\lambda)\, in\, t_2 = \{t_2 : (1, 2, 3)|t_2 : (7, 8, 10)\}$.*

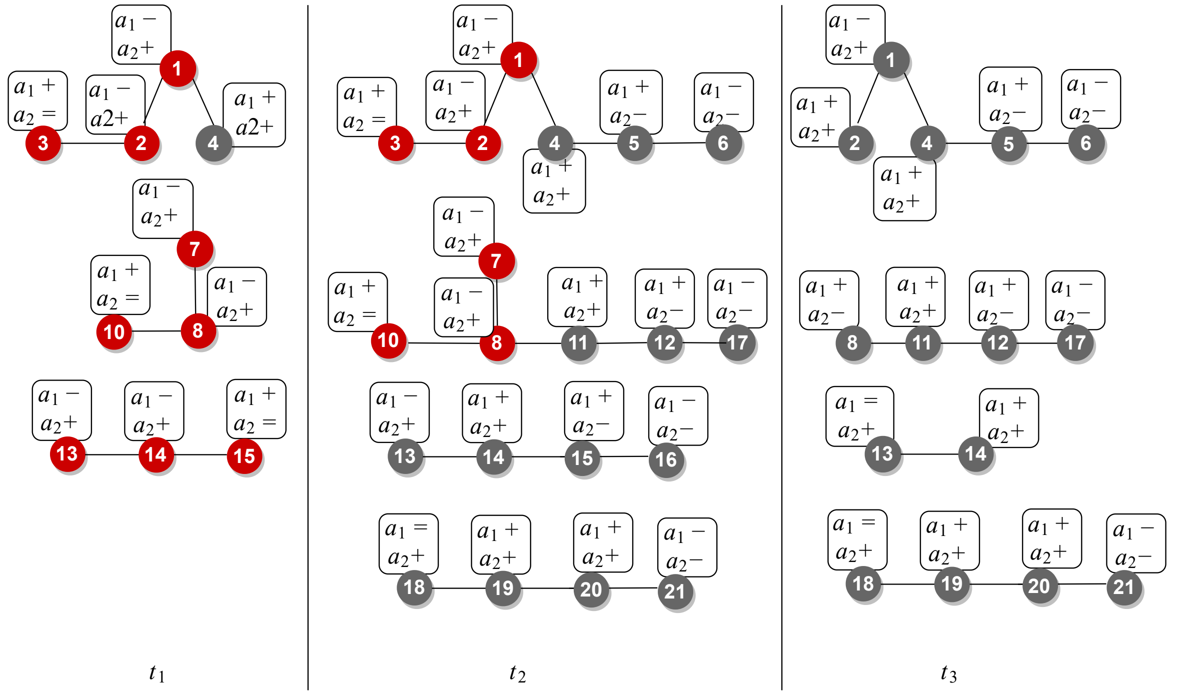


Figure 4.14: Frequent subgraph

Definition 25 (frequent subgraph)**.** *A subgraph of $\mathcal{G}$ is defined as a set of attribute values λ and its occurrences $Occurrence(\lambda)\, in\, T'$. It is denoted as $\langle \lambda, Occurrence(\lambda)\, in\, T' \rangle$. A subgraph is frequent in $\mathcal{G}$ if the set of attribute values λ appears more than once (i) in time, i.e., it has occurrences at least in two different timestamps in T' and (ii) in space, i.e., it has at least two occurrences (or two groups of connected vertices) for each timestamp in T'.*

Example 12. *Consider the subgraph $\langle (a_1 - a_2+, a_1 - a_2+, a_1 + a_2 =), \{t_1 : (1,2,3) | t_1 : (7,8,10) | t_1 : (13,14,15) | t_2 : (1,2,3) | t_2 : (7,8,10)\} \rangle$ represented by the red connected vertices in Fig. 4.14. The subgraph is frequent in time as it has occurrences at two different timestamps, t_1 and t_2. Moreover, it is also frequent in space, as it includes more than one group of connected vertices for the same timestamp. For instance, in t_1, the subgraph has three occurrences: $(1,2,3), (7,8,10), (13,14,15)$. So the spatio-temporal frequency of this subgraph is 5 because it has 5 occurrences.*

Definition 26 (frequent sequential subgraph evolution)**.** *A frequent sequential subgraph evolution (FSSE) in $\mathcal{G}$ is a sequence of time-ordered frequent subgraphs representing the evolution mechanism of several groups of connected vertices. It is denoted as follows :*

$$P = \langle \{\lambda_1; ...; \lambda_n\}, \{T_1 : Occurrence_1(\lambda_1); \ldots; Occurrence_1(\lambda_n) | \ldots \\ |T_k : Occurrence_k(\lambda_1); \ldots; Occurrence_k(\lambda_n)\} \rangle$$

where :

- *$\{\lambda_1; ...; \lambda_n\}$ represents a sequence of attribute values sets separated by semicolons;*
- *$\{T_1 : Occurrence_1(\lambda_1); \ldots; Occurrence_1(\lambda_n), | \ldots | T_k : Occurrence_k(\lambda_1); \ldots ; Occurrence_k(\lambda_n)\}$ represents the occurrences of the sequence $\{\lambda_1; ...; \lambda_n\}$ at the different time intervals $\{T_1, \ldots, T_k\}$. Occurrences are separated by vertical bars;*

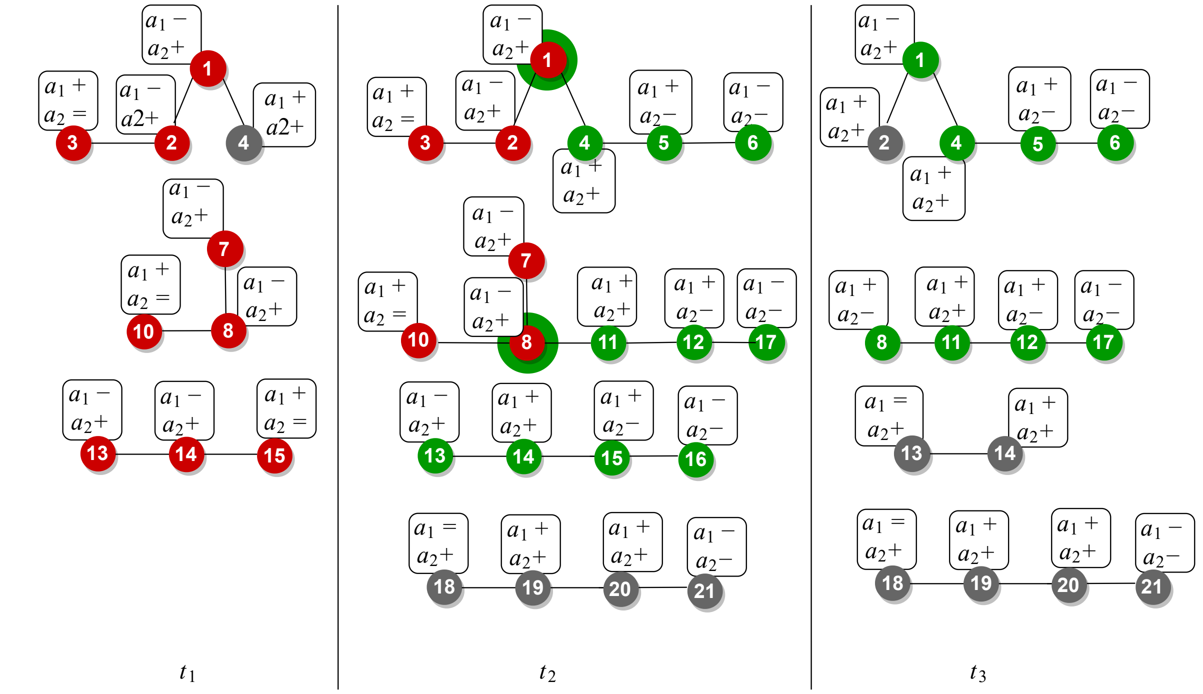


Figure 4.15: A frequent sequential subgraph evolution

- *Each $T_i : Occurrence_i(\lambda_1); ..., Occurrence_i(\lambda_n)$ represents the occurrence of the sequence $\{\lambda_1; ...; \lambda_n\}$ at the time interval T_i where, for $T_i = \{t_i, ..., t_j\}, 1 \le i \le j \le |T|$, t_i the starting time and t_j represents the end time of the sequence of occurrences;*

- *Each occurrence of the sequence should have at least one common vertex to represent the evolution mechanism of a common core of connected vertices.*

Example 13. *Consider the frequent subgraph $\langle(a_1 - a_2+, a_1 - a_2+, a_1 + a_2 =), \{t_1 : (1, 2, 3)|t_1 : (7, 8, 10)|t_1 : (13, 14, 15)|t_2 : (1, 2, 3)|t_2 : (7, 8, 10)\}\rangle$ in red, in Fig. 4.14. If we create a sequence, it should be followed by a frequent subgraph during the time interval $[t_2, t_3]$. As shown in Fig. 4.15, the green subgraph $\langle\{(a1 - a2+, a1 + a2+, a1 + a2-, a1 - a2-)\}, \{t_2 : (1, 4, 5, 6)|t_2 : (8, 11, 12, 17)|t_2 : (13, 14, 15, 16)|t_3 : (1, 4, 5, 6)|t_3 : (8, 11, 12, 17)\}\rangle$ is frequent during $[t_2, t_3]$. Moreover, each of its occurrences has at least one common vertex with the red subgraph. The red occurrence $(1, 2, 3)$ has the vertex 1 in common with the green occurrence $(1, 4, 5, 6)$. The red occurrence $(7, 8, 10)$ has the vertex 8 in common with the green occurrence $(8, 11, 12, 17)$. The red occurrence $(13, 14, 15)$ has the vertices 13, 14 and 15 in common with the green occurrence $(13, 14, 15, 16)$. Therefore, we can create the sequence of subgraphs $\langle\{(a1 - a2+, a1 - a2+, a1 + a2 =); (a1 - a2+, a1 + a2+, a1 + a2-, a1 - a2-)\}, \{t_1, t_2 : (1, 2, 3); (1, 4, 5, 6)|t_1, t_2 : (7, 8, 10); (8, 11, 12, 17)|t_1, t_2 : (13, 14, 15); (13, 14, 15, 16)|t_2, t_3 : (1, 2, 3); (1, 4, 5, 6)|t_2, t_3 : (7, 8, 10); (8, 11, 12, 17)\}\rangle$ from t_1 to t_3. It is a FSSE pattern of size 2, i.e., composed of two frequent subgraphs: (i) a frequent subgraph in $\{t_1, t_2\}$ (in red) and (ii) a frequent subgraph in $\{t_2, t_3\}$ (in green). The spatio-temporal frequency of this pattern is 5 since it has 5 occurrences.*

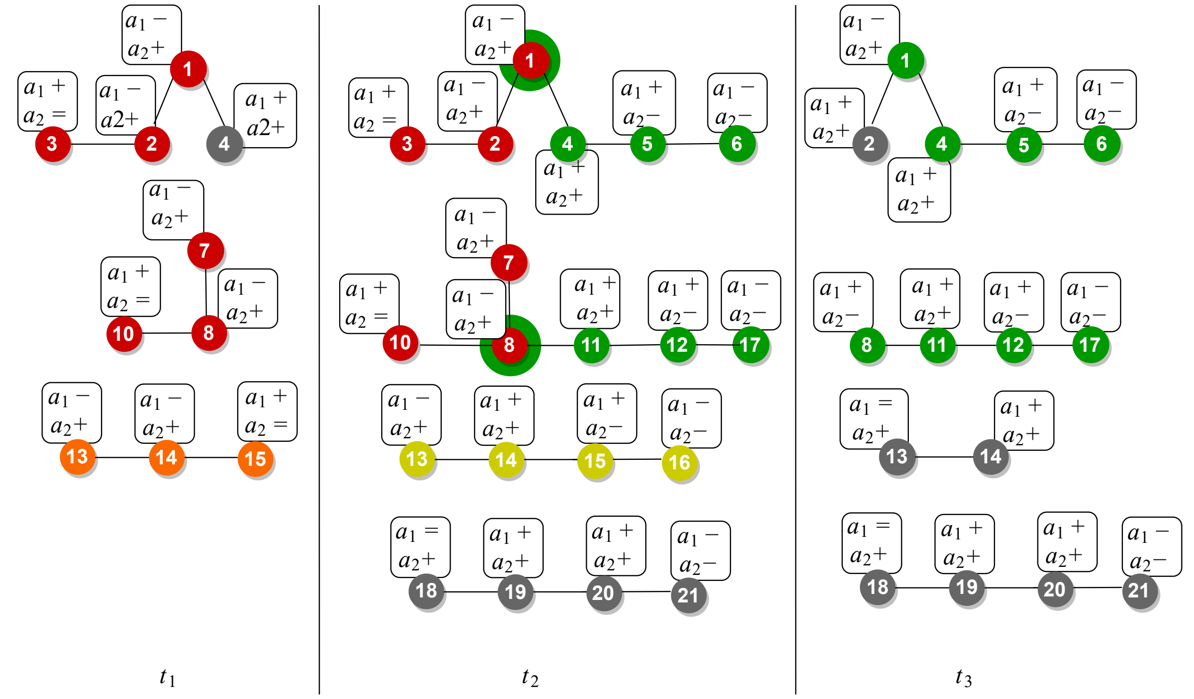
A FSSE pattern meets the challenges posed in previous sections. First, it is a **representative** evolution because it accounts for the evolution of several groups of vertices. In Example 13, we observe that the FSSE pattern is over several groups of vertices, such as the group $(1, 2, 3)$ or $(7, 8, 10)$. This is possible because a FSSE pattern counts the frequency of occurrences in time and space instead of only in time. Second, a FSSE pattern is a **sequential** evolution because it is a time-ordered sequence of subgraphs. Finally, it is a **complete** evolution because it captures the evolution of attribute values of vertices through the sequence of attribute values sets. In Example 13, we observe that the FSSE pattern describes the changes in the attribute trends of groups of vertices. Moreover, it captures the evolution of graph topology by simply imposing a constraint of connectivity within the groups of vertices. This implies that between two timestamps, the configuration of a group of vertices can change (addition and removal of vertices or edges) as long as there is a connectivity between vertices in the new configuration. In Example 13, we observe for instance that in the occurrence $t_1, t_2 : (13, 14, 15); (13, 14, 15, 16)$ of the FSSE, the configuration of the group of vertices $(13, 14, 15)$ has changed from t_1 to t_2. A new vertex 16 has been added to the initial group of vertices.

We compare the FSSE and the closest pattern: recurrent patterns (Cheng et al., 2017). Recurrent patterns are sequential and complete, but not representative. The example in Fig. 4.16 illustrates this difference. With a frequency of 2, two recurrent patterns are extracted. The first recurrent pattern is the group of connected vertices in red, $(1 : a_1 - a_2+, 2 : a_1 - a_2+, 3 : a_1 + a_2 =)$, which is followed by the group of connected vertices in green, $(1 : a_1 - a_2+, 4 : a_1 + a_2+, 5 : a_1 + a_2-, 6 : a_1 - a_2-)$. The frequency of this pattern is 2, as it appears twice over time: the first time from t_1 to t_2 and the second time from t_2 to t_3. The second recurrent pattern is a group of connected vertices in red, $(7 : a_1 - a_2+, 8 : a_1 - a_2+, 10 : a_1 + a_2 =)$, which is followed by another group of connected vertices in green, $(8 : a_1 - a_2+, 11 : a_1 + a_2+, 12 : a_1 + a_2-, 17 : a_1 - a_2-)$, with a frequency of 2. We notice another sequence of subgraphs composed by the orange group of connected vertices $(13 : a_1 - a_2+, 14 : a_1 - a_2+, 15 : a_1 + a_2 =)$ which is followed by the yellow group of connected vertices $(13 : a_1 - a_2+, 14 : a_1 + a_2+, 15 : a_1 + a_2-, 16 : a_1-, a_2-)$, representing exactly the same attribute trend evolution as the two patterns extracted above. However, it is not considered as a recurrent pattern since its temporal frequency is 1. In comparison, one extracted frequent sequential subgraph evolution is the sequence $(a_1 - a_2+, a_1 - a_2+, a_1 + a_2 =); (a_1 - a_2+, a_1 + a_2+, a_1 + a_2-)$ for several groups of connected vertices. It groups all the frequent and infrequent recurrent patterns to generate a much more general pattern. The frequency of this FSSE pattern is considered in one more dimension, the spatial one. So, its spatio-temporal frequency is 5.

3.3 Complementary Constraints

Let $P = \langle \{\lambda_1; ...; \lambda_n\}, \{T_1 : Occurrence_1(\lambda_1); \ldots; Occurrence_1(\lambda_n) \,|\, \ldots |T_k : Occurrence_k(\lambda_1); \ldots; Occurrence_k(\lambda_n)\}\rangle$ be a FSSE pattern. Several constraints are defined to complete the extraction of this pattern in a dynamic attributed graph for two purposes. The first objective is to let the users precise even more certain aspects of the pattern via a set of constraints. Indeed, different users may have different requirements or perspectives on what constitutes an interesting pattern. The second objective is to reduce the search space and improve the efficiency of the mining algorithm.

Definition 27 (connectivity)**.** *Vertices of a graph represent entities, and edges represent relationships between these entities. During pattern extraction, vertices must be connected by edges to extract potentially correlated evolutions among a set of objects. In Fig. 4.15, the pattern* $(a_1 - a_2+, a_1 - a_2+, a_1 + a_2 =); (a_1 - a_2+, a_1 + a_2+, a_1 + a_2-, a_1 - a_2-)$ *occurs on a sequence of groups of connected vertices, such as* $(1, 2, 3); (1, 4, 5, 6)$ *from* t_1 *to* t_2 *and*



first extracted recurrent pattern with a temporal frequency of 2:
$(1 : a_1 - a_2+, 2 : a_1 - a_2+, 3 : a_1 + a_2 =)(1 : a_1 - a_2+, 4 : a_1 + a_2+, 5 : a_1 + a_2-, 6 : a_1 - a_2-)$

second extracted recurrent pattern with a temporal frequency of 2:
$(7 : a_1 - a_2+, 8 : a_1 - a_2+, 10 : a_1 + a_2 =)(8 : a_1 - a_2+, 11 : a_1 + a_2+, 12 : a_1 + a_2-, 17 : a_1 - a_2-)$

an extracted frequent sequential subgraph evolution pattern with a spatio-temporal frequency of 5:

$(a_1 - a_2+, a_1 - a_2+, a_1 + a_2 =); (a_1 - a_2+, a_1 + a_2+, a_1 + a_2-, a_1 - a_2-)$
3 *Occurrences in* t_1, t_2 : $(1,2,3); (1,4,5,6)$ *and* $(7,8,10); (8,11,12,17)$ *and* $(13,14,15); (13,14,15,16)$
2 *Occurrences in* t_2, t_3 : $(1,2,3); (1,4,5,6)$ *and* $(7,8,10), (8,11,12,17)$

Figure 4.16: FSSE VS reccurrent patterns

$(7, 8, 10); (8, 11, 12, 17)$ *from* t_2 *to* t_3.

Definition 28 (volume)**.** *The volume refers to the number of vertices of a subgraph. Let* $vol(P) = \min_{\forall i \in [1,n]} |\lambda_i|$ *be the volume of a pattern* P*. A pattern* P *is sufficiently voluminous if and only if* $vol(P) \geq minvol$*, where* ***minvol*** *is a minimum number of vertices of a subgraph. The minvol constraint should be at least equal to 2 to be a subgraph. The user can also define the maximum number of vertices of a subgraph, denoted as* ***maxvol****, such as* $vol(P) \leq maxvol$*. For example, the pattern* $(a_1 - a_2+, a_1 - a_2+, a_1 + a_2 =); (a_1 - a_2+, a_1 + a_2+, a_1 + a_2-, a_1 - a_2-)$ *has a volume of 3.*

Definition 29 (temporal continuity)**.** *An evolution may include different vertices at each timestamp. However, it is difficult for end users to interpret the evolution of vertices without a direct relation between them at each step. Hence, it is desirable to study evolution around a common core of vertices. To do so, a constraint is defined to follow a minimum number of common vertices over time, denoted as* ***mincom****. Let denote* $Occurrence_j(P)$ *is a* j^{th} *instance of pattern* P *and* $com(Occurrence_j(P)) = |\cap_{\forall i \in 1,\ldots,n} Occurrence_j(\lambda_i)|$ *be the common number of vertices occurring in the instance sequence* j*.* P *is a continuous pattern iff* $\forall j \in \{1, .., k\}$ $com(Occurrence_j(P)) \geq mincom$*. Consider* P *the pattern*

$(a_1 - a_2+, a_1 - a_2+, a_1 + a_2 =); (a_1 - a_2+, a_1 + a_2+, a_1 + a_2-, a_1 - a_2-)$ in Fig 4.15. All instances of the pattern P have at least one common vertex. For instance, the subgraphs of the occurrence $(7, 8, 10); (8, 11, 12, 17)$, at t_1 and t_2, have one common vertex, which is 8.

Definition 30 (spatio-temporal frequency)**.** *The frequency constraint, denoted as **minsup**, is a user-defined threshold to filter patterns which occur more than a minimum number in time and in space. The frequency of P is the number of occurrences of the pattern P, $sup(P) = k$. Consequently, P is a frequent evolution iff $sup(P) \geq minsup$. For example, in Fig. 4.15, the frequency of the sequence $(a_1 - a_2+, a_1 - a_2+, a_1 + a_2 =); (a_1 - a_2+, a_1 + a_2+, a_1 + a_2-, a_1 - a_2-)$ is 5, as the pattern appears 5 times.*

3.4 Problem Setting

Given a dynamic attributed graph $\mathcal{G}$, the problem is to extract the complete set of frequent sequential subgraph evolutions in $\mathcal{G}$, denoted as Sol, such that $\forall P \in Sol$, 1) P is frequent (i.e., $sup(P) \geq minsup$); 2) the occurrences of P are connected at each time; 3) P is sufficiently voluminous (i.e., $minvol \leq vol(P) \leq maxvol$); 4) P is centered around a core of vertices sufficiently large (i.e., $com(P) \geq mincom$), where $minvol, maxvol, minsup$ and $mincom$ are user-defined thresholds.

4 Mining Frequent Sequential Subgraph Evolutions (FSSEMiner Algorithm)

In this section, we present a novel mining algorithm: *FSSEMiner*. FSSEMiner allows for mining a novel pattern, FSSE. FSSEMiner is based on a novel mining strategy, the *graph addition*, to reduce the need to traverse the entire search space. This strategy allows constructing frequent subgraphs by a linear addition operation of the candidate subgraphs. Moreover, FSSEMiner uses the constraints defined in the previous section to reduce the search space. In the following sections, we first present the FSSEMiner algorithm (Section 4.1). Then, we explain in detail the process of the algorithm, including the *graph addition* (Section 4.2). Finally, we calculate the time complexity of the algorithm (Section 4.3).

4.1 Overview of the Algorithm

The FSSEMiner algorithm is presented in Algorithm 5 and its process is illustrated in Fig. 4.17. *Line 1* corresponds to the extraction of subgraph candidates, satisfying the volume constraints ($minvol$ and $maxvol$). *Line 3-6* generates size-1 patterns starting at t_1 and satisfying the frequency constraint ($minsup$). To do so, the algorithm first constructs all time combinations including t_1 (T_1^k, *line 4*). It generates size-1 patterns P_i by performing additions of subgraphs occurring at these times ($Union$ function, *line 5*). After that, the other times are processed incrementally. Then, the algorithm constructs all time combinations, including t_i (T_i^k, *line 9*) and extracts size-1 patterns P_i from the additions of subgraphs (lines 10-11). Then, it extends each pattern P generated from the previous iteration with these size-1 patterns to create a sequence of subgraphs (*lines 12-13*). If the pattern P', resulting from the extension of P with P_i, satisfies the temporal continuity ($mincom$) and frequency constraints ($minsup$), it is added to the set of patterns

```
Algorithm 5: $FSSEMiner$ : Mining frequent sequential subgraph evolutions
   Input: $\mathcal{G}$ : a dynamic attributed graph , $minsup$, $minvol$, $maxvol$, $mincom$
   Output: $Sol$: set of frequent sequential subgraph evolutions satisfying the constraints
   /* Step 1: Get the occurrences of the subgraph candidates of $\mathcal{G}$ */
1  $S \leftarrow ExtractionSubgraphs(\mathcal{A}, \mathbb{D}, minvol, maxvol)$ /* see Algorithm 6 */
   /* or $S = \{S_i$ set of subgraphs of $G_t, t \in T \mid \forall s_i \in S_i, s_i = \langle\lambda_i, Occurrence(\lambda_i)\, in\, t\rangle, minvol \leq |Occurrence(\lambda_i)| \leq maxvol\}$ */
2  $Cand_i \leftarrow \emptyset, \forall i \in \{1, 2, ..., |\mathcal{T}|\}$
   /* Step 2: Generate size-1 patterns */
3  for $k = 1$ to $|\mathcal{T}|$ do
4      for each $T_1^k \subseteq \mathcal{T}$ such as $t_1 \in T_1^k$ do
           /* time combination including $t_1$ of size $k$ */
5          $P_{union} \leftarrow Union(S, T_1^k, minsup)$ /* see Algorithm 9 */
           /* or $P_{union} = \{S_{union}$ set of frequent subgraphs in $T_1^k, \mid \forall s_{union} \in S_{union}, s_{union} = (\lambda, Occurrence_{union}), Occurrence_{union} = \cup_{\forall t \in T_1^k} Occurrence(\lambda)\, in\, t$ and $|Occurrence_{union}| \geq minsup\}$ */
6          $Cand_1 \leftarrow Cand_1 \cup P_{union}$
   /* Step 3: Extension of patterns */
7  $Sol_i = \emptyset, \forall i \in \{1, 2, ..., |\mathcal{T}|\}$
8  for $i = 2$ to $|\mathcal{T}|$ do
9      for each $T_i^k \subseteq \mathcal{T}$ such as $t_i \in T_i^k$ do
10         $P_{union} \leftarrow Union(S, T_i^k, minsup)$
           /* or $P_{union} = \{S_{union}$ set of frequent subgraphs in $T_i^k, \mid \forall s_{union} \in S_{union}, s_{union} = \langle\lambda, Occurrence_{union}\rangle, Occurrence_{union} = \cup_{\forall t \in T_1^k} Occurrence(\lambda)\, in\, t$ and $|Occurrence_{union}| \geq minsup\}$ */
11         for each $P_i \in P_{union}$ do
12             for each $P$ such as $P \in Cand_{i-1}$ do
13                 $P' \leftarrow ExtendWith(P, P_i)$
14                 if $com(P') \geq mincom$ and $|P'| \geq minsup$ then
15                     $Cand_i \leftarrow Cand_i \cup \{P'\}$
16                 else
17                     $Sol_{i-1} \leftarrow Sol_{i-1} \cup \{P\}$
18                     $Cand_i \leftarrow Cand_i \cup \{P_i\}$
19 $Sol = MergeUpdate(\bigcup_{\forall i \in \mathcal{T}} Sol_i)$
20 Return $Sol$
```

generated at time t_i (*lines 14-15*). Otherwise, P is added to the set of solutions, and P_i is saved for future extensions (*lines 17-18*). Finally, all solutions generated at each time are put together, and associated times are updated (*line 19*).

4.2 Process of the Algorithm

4.2.1 Extraction of Subgraph Candidates

The first step of the FSSEMiner algorithm process is to extract all possible candidate subgraphs in a dynamic attribute graph $\mathcal{G}$. Algorithm 6 describes the generation of the set of these candidate subgraphs.

First, the algorithm creates the set of all possible attribute value combinations, denoted as $Cand_{\mathcal{A}\mathbb{D}}$, using the function $CombineAttributeValue$ (*line 1* and detailed in Algorithm 7).

Second, it creates the set of all possible subgraphs, denoted as $Cand_\lambda$, which represents subgraphs having a number of vertices included in $[minvol, maxvol]$ and whose vertices

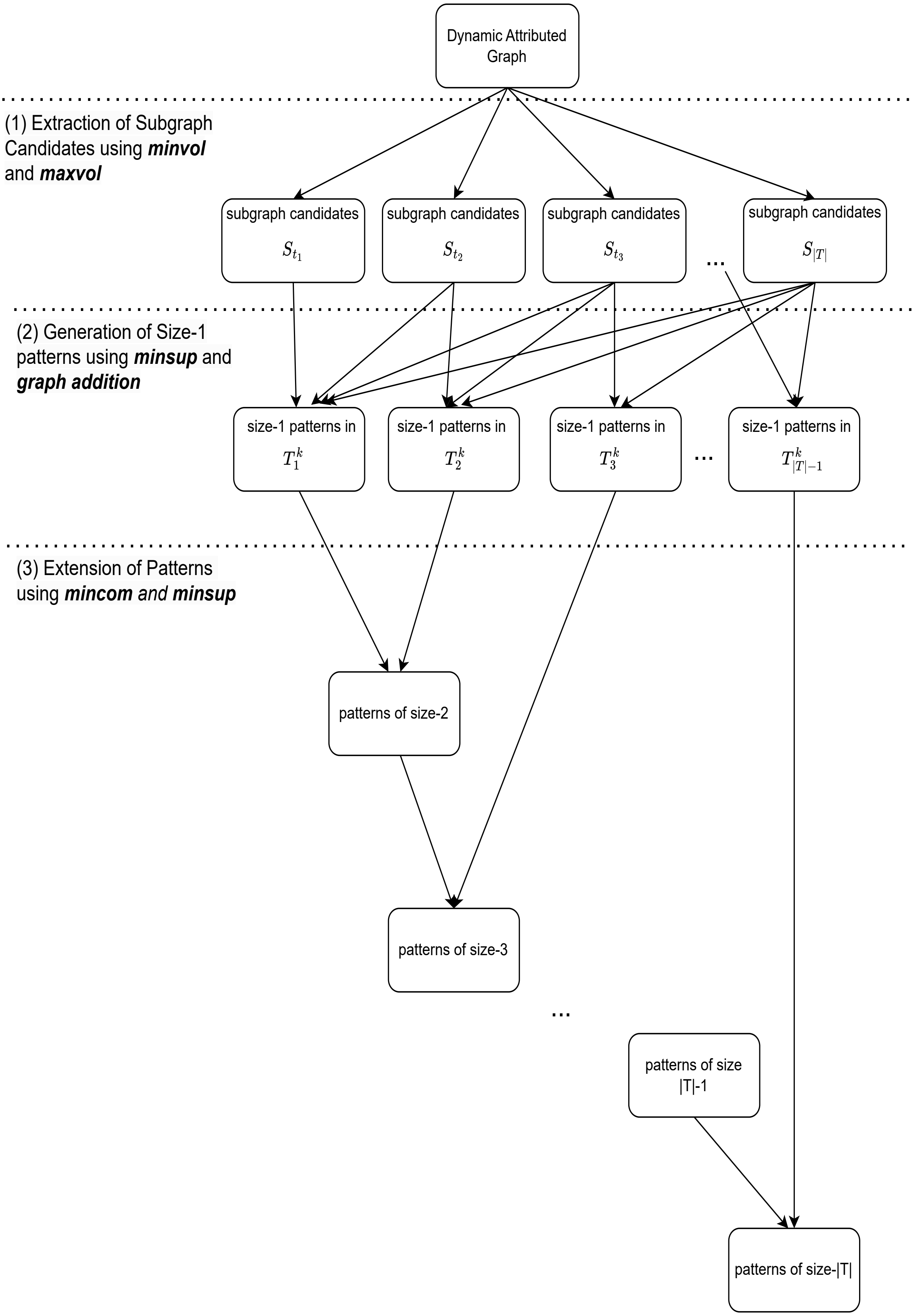

Dynamic Attributed Graph
(1) Extraction of Subgraph Candidates using minvol and maxvol
subgraph candidates S_{t_1}
subgraph candidates S_{t_2}
subgraph candidates S_{t_3}
...
subgraph candidates $S_{|T|}$
(2) Generation of Size-1 patterns using minsup and graph addition
size-1 patterns in T_1^k
size-1 patterns in T_2^k
size-1 patterns in T_3^k
...
size-1 patterns in $T_{|T|-1}^k$
(3) Extension of Patterns using mincom and minsup
patterns of size-2
patterns of size-3
...
patterns of size |T|-1
patterns of size-|T|

Figure 4.17: Main process of the FSSE algorithm

Algorithm 6: *ExtractionSubgraphs*: Function to compute all combinations of subgraphs

```
Input: an attribute set: $\mathcal{A}$, a set of value domains: $\mathbb{D}$, a minimum volume of subgraphs: $minvol$, a maximum
       volume of subgraphs: $maxvol$
Output: a set of subgraphs: $S$
1  $Cand_{\mathcal{A}\mathbb{D}} \leftarrow \cup_{\forall k \in [1,|\mathcal{A}|]} CombineAttributeValue(\mathcal{A}, \mathbb{D}, k)$ /* see $CombineAttributeValue$ function */
2  $Cand_{\lambda} \leftarrow \emptyset$ /* to record the candidate subgraphs */
   /* create the candidate subgraphs with the attribute value sets in $Cand_{\mathcal{A}\mathbb{D}}$ for
      each volume between $minvol$ and $maxvol$ */
3  foreach $vol \in [minvol, maxvol]$ do
4      $Cand_{\lambda,vol} \leftarrow \emptyset$ /* to record the candidate subgraphs for the volume $vol$ */
       /* Create subgraphs with a single vertex */
5      foreach $\lambda_i \in Cand_{\mathcal{A}\mathbb{D}}$ do
6          $Cand_{\lambda,vol} \leftarrow Cand_{\lambda,vol} \cup \{(\lambda_i)\}$
       /* Extend subgraphs with another vertex */
7      $m = minvol$
8      while $m \leq vol$ do
9          $NewCand_{\lambda,vol} \leftarrow \emptyset$
10         foreach $\lambda_i \in Cand_{\lambda,vol}$ do
11             foreach $\lambda_j \in Cand_{\mathcal{A}\mathbb{D}}$ do
12                 $NewCand_{\lambda,vol} \leftarrow NewCand_{\lambda,vol} \cup \{(\lambda_i, \lambda_j)\}$
13         $Cand_{\lambda,vol} \leftarrow NewCand_{\lambda,vol}$
14         $m = m + 1$
15     $Cand_{\lambda} \leftarrow Cand_{\lambda} \cup Cand_{\lambda,vol}$
   /* get the occurrences of each attribute value set in the graph */
16 $S \leftarrow \emptyset$ /* to record final subgraphs */
17 $S_t \leftarrow \emptyset, \forall t \in T$ /* to record final subgraphs */
18 foreach $t \in T$ do
19     foreach $\lambda_i \in Cand_{\lambda}$ do
20         $S_t \leftarrow S_t \cup \langle \lambda_i, Occurrence(\lambda_i)\, in\, t \rangle$ /* see details of the function $Occurrence$ */
           /* Each $S_t \in S$ is ordered in the same way according to $\lambda$ */
21 $S \leftarrow S \cup S_t$
22 Return $S$
```

contain the attribute value sets in $Cand_{\mathcal{A}\mathbb{D}}$ *(line 2-15).*

Finally, for each generated subgraph $\lambda \in Cand_{\lambda}$, a Depth-First Search (DFS) strategy is used to compute its occurrences $Occurrence(\lambda_i) \in t$ in each $G_t \in \mathcal{G}$ *(line 18-22).* The function *Occurrence* is detailed in Algorithm 8. The anti-monotonicity property is respected to find anti-monotonic subgraphs Fiedler and Borgelt (2007). The result is the set of subgraphs satisfying the *volume* and *connectivity* constraints and denoted as $S = \{S_t \text{ set of subgraphs of } G_t, t \in T \mid \forall s_i \in S_t, s_i = \langle \lambda_i, Occurrence(\lambda_i)\, in\, t \rangle, minvol \leq |Occurrence(\lambda_i)| \leq maxvol\}$.

Example 14. *Let us consider the dynamic attributed graph in Fig 4.13. We have* $\mathcal{A} = \{a_1, a_2\}$ *and* $\mathbb{D}_{a_i} = \{+, -, =\}, \forall a_i \in \mathcal{A}$*. At the first step of Algorithm 6, we obtain the set of attribute values* $Cand_{\mathcal{A}\mathbb{D}} = \{a_1+, a_1-, a_2+, a_2-, a_1+a_2+, a_1+a_2-, a_1-a_2+, a_1-a_2-\}$ *using the CombineAttributeValue function (Algorithm 7). Consider that* $minvol = 2$ *and* $maxvol = 3$*. Then, subgraphs with a volume of* 2 *(i.e., composed by two vertices) and* 3 *(i.e., composed by three vertices) based on* $Cand_{\mathcal{A}\mathbb{D}}$ *are created. We obtain, for instance, a subgraph of volume* 2 $\lambda_1 = (a_1 - a_2+, a1 - a_2+)$ *and a subgraph of volume* 3 $\lambda_2 = (a_1 - a_2+, a1 - a_2+, a1 + a2 =)$*. Finally, the algorithm searches for the occurrences of each* λ_i *for each timestamp* $t \in T$ *in the dynamic attributed graph. We obtain for*

Algorithm 7: *CombineAttributeValue*: Function to make attribute-value combination of size k

```
Input: an attribute set: $\mathcal{A} = \{a_1, ...a_i, ...a_{|\mathcal{A}|}\}$, a set of value domains: $\mathbb{D} = \{\mathbb{D}_{a_1}, ...\mathbb{D}_{a_i}, ...\mathbb{D}_{|\mathcal{A}|}\}$, a size of
       combinations $k$
Output: a set of attribute value combinations: $Cand_{\mathcal{A}\mathbb{D}}$
$Cand_{\mathcal{A}\mathbb{D}} \leftarrow \emptyset$
/* Create size-1 combination of attribute value */
for $i = 1$ to $|\mathcal{A}| - k + 1$ do
    for $j = 1$ to $|\mathbb{D}_{a_i}|$ /* where $\mathbb{D}_{a_i} \in \mathbb{D}$ */
     do
        $Cand_{\mathcal{A}\mathbb{D}} \leftarrow \{a_i \cdot d_j\}$ /* where $a_i \in \mathcal{A}$ and $d_j \in \mathbb{D}_{a_i}$ */
/* Extend the combination if $k \geq 2$ */
$i = 2$
while $i \leq k$ do
    /* Extend the combination */
    $NewCand_{\mathcal{A}\mathbb{D}} \leftarrow \emptyset$
    foreach $cand \in Cand_{\mathcal{A}\mathbb{D}}$ do
        for $j = 1$ to $|\mathbb{D}_{a_i}|$ /* where $\mathbb{D}_{a_i} \in \mathbb{D}$ */
         do
            $NewCand_{\mathcal{A}\mathbb{D}} \leftarrow \{cand \cdot a_i \cdot d_j\}$/* where $a_i \in \mathcal{A}$ and $d_j \in \mathbb{D}_{a_i}$ */
    $Cand_{\mathcal{A}\mathbb{D}} \leftarrow NewCand_{\mathcal{A}\mathbb{D}}$
    $i = i + 1$
Return $Cand_{\mathcal{A}\mathbb{D}}$
```

λ_1 at t_1, $Occurrence(\lambda_1)\, in\, t_1 = \{t_1 : (1,2)|t_1 : (7,8)|t_1 : (13,14)\}$ and for λ_2 at t_1, $Occurrence(\lambda_2)\, in\, t_1 = \{t_1 : (1,2,3)|t_1 : (7,8,10)|t_1 : (13,14,15)\}$.

4.2.2 Generation of Size-1 Patterns by Graph Addition

The construction of size-1 patterns is the fundamental building block for constructing the final patterns. To generate patterns of size-1, the *graph addition* strategy is proposed. It is a linear addition operation of candidate subgraphs to construct frequent subgraphs (the *minsup* constraint is verified). More precisely, this consists in making the union of the occurrences for the same pattern (i.e., same attribute values λ) at different times (Fig 4.18). This strategy avoids graph traversal operations, which would be exponential using existing strategies.

Algorithm 9 describes the graph addition process. Let be n timestamps $t_i, ..., t_j \in T$, where $1 \leq n \leq |T|$ and $1 \leq i < j \leq |T|$. The addition of n subgraphs $s_i = \langle \lambda_i, Occurrence(\lambda_i)\, in\, t_i \rangle$,..., $s_j = \langle \lambda_j, Occurrence(\lambda_j)\, in\, t_j \rangle$ is denoted as $s_{union} = \langle \lambda, Occurrence_{union}(\lambda) \rangle$ where $\lambda = \lambda_i = ... = \lambda_j$ and $Occurrence_{union}(\lambda) = Occurrence(\lambda_i)\, in\, t_i \cup ... \cup Occurrence(\lambda_j)\, in\, t_j$ (*line 3*). s_{union} is a subgraph composed of the union of occurrences of the n initial subgraphs having the same pattern. If $|Occurrence_{union}(\lambda)| \geq minsup$, the algorithm keeps s_{union} in the mining process (*lines 5-6*). For the special case where $n = 1$ and $i = j$, the result of the addition of a subgraph is itself. This case is necessary because a size-1 pattern (one subgraph) could also be spatially frequent in one timestamp.

Example 15. *Let us suppose that $minsup = 4$. In Fig. 4.13, The subgraphs $s_1 = \langle \{(a1-a2+, a1-a2+, a1+a2=)\}, \{t_1 : (1,2,3)|t_1 : (7,8,10)|t_1 : (13,14,15)\}$ in t_1 and $s_2 = \langle \{(a1-a2+, a1-a2+, a1+a2=)\}, \{t_2 : (1,2,3)|t_2 : (7,8,10)\}$ in t_2 have the same pattern. By adding s_1 and s_2, the pattern $s_{union} = \langle \{(a1-a2+, a1-a2+, a1+a2 =$*

Algorithm 8: *Occurrence*: Function to get the set of occurrences of a subgraph at a given timestamp

```
Input: a timestamp : $t$, vertices at timestamp $t$: $V_t$, edges at timestamp $t$: $E_t$, a set of attribute values of a
       subgraph: $\lambda = (\lambda_1, ..., \lambda_i, ..., \lambda_{|\lambda|})$
Output: Set of sets of occurrences: $OccFinal$
/* Step 1 :  Get the occurrences of the subgraph                                    */
$Occ \leftarrow \emptyset$ /* to record candidate occurrences of the subgraph               */
$CandV \leftarrow V_t$ /* to record candidate vertices of occurrences                      */
foreach $v \in CandV$ do
    i = 1
    /* index to traverse the attribute-value sets of the subgraph                   */
    if $\lambda_i \subseteq \lambda_t(v) \mid \lambda_i \in \lambda$ then
        /* check if the first attribute-value set of the subgraph is included in
           the attribute-value set of the vertex $v$                               */
        /* create a vector of occurrences (or connected vertices)                   */
        $newocc \leftarrow (v)$
        i = i + 1
        for each neighbor of $v$ such as $N(v) \in CandV$ and $(v, N(v)) \in E_t$ do
            /* check if the next attribute-value set of the subgraph is equal to
               the attribute-value set of the neighbor of the vertex $v$            */
            if $\lambda_i \subseteq \lambda(N(v)) \mid \lambda_i \in \lambda$ then
                $newocc \leftarrow (newocc, N(v))$
                if $i < |\lambda|$ then
                    i = i + 1
                else if  $i = |\lambda|$ then
                    $Occ \leftarrow Occ \cup \{newocc\}$
                    Exit For
            else
                Exit For
    $CandV \leftarrow CandV \setminus \{v\}$
    /* not traverse the vertex $v$ anymore                                          */
/* Step 2:  Apply the anti-monotonicity property of subgraphs                         */
/* Step 2.1.   :  Compute the overlapping occurrences of the subgraph                 */
$MapV \leftarrow \emptyset$
$Overlap \leftarrow \emptyset$
for $i = 1$ to $|\lambda|$ do
    $Overlap_i \leftarrow \emptyset$
    /* $Overlap_i$ is the set of occurrences overlapping on identical vertices at
       the position $i$                                                             */
    $MapV_i \leftarrow \emptyset$
    /* $MapV_i$ is the set of vertices of all occurrences at the position $i$        */
    foreach $occ_j \in Occ$ do
        $MapV_i \leftarrow MapV_i \cup \{v_i | v_i \in occ_j\}$
        /* $v_i$ is the vertex at the $i$ position of the occurrence $occ_j$         */
        $newoverlap \leftarrow \emptyset$ /* to record a set of occurrences having the same vertex at
           the position $i$.                                                        */
        foreach $occ_k \in Occ \setminus \{occ_j\}$ do
            if $v_i \in occ_j = w_i \in occ_k$ then
                /* $w_i$ is the vertex at the $i$ position in the $occ_k$ vector     */
                $newoverlap \leftarrow newoverlap \cup \{occ_j, occ_k\}$
        $Overlap_i \leftarrow Overlap_i \cup \{newoverlap\}$
        /* $Overlap_i$ is a set of occurrence sets.                                 */
    $MapV \leftarrow MapV \cup \{MapV_i\}$ /* $MapV$ is a set of vertex sets.                   */
    $Overlap \leftarrow Overlap \cup \{Overlap_i\}$ /* $Overlap$ is a set of sets of occurrence sets.   */
```

```
/* Step 2.2: Compute the final set of occurrences of the subgraph */
$sup \leftarrow min|MapV_i| \mid MapV_i \in MapV$ /* $sup$ is the frequency of the subgraph */
$OccFinal \leftarrow \emptyset$
if $\forall MapV_i \in MapV, |MapV_i| = sup$ then
    if $\forall Overlap_i \in Overlap, Overlap_i = \emptyset$ then
        $OccFinal \leftarrow Occ$
else
    foreach $i \in [1, |MapV|]$ do
        if $|MapV_i| = sup$ then
            $NonOverlap \leftarrow Occ \setminus \{\cup_{j \in [1,|Overlap_i|]} OccSet_j \in Overlap_i \in Overlap\}$
            $OverlapComb \leftarrow \emptyset$ /* to record the set of combinations of overlapping occurrences */
            foreach $occ \in OccSet_1 \mid OccSet_1 \in Overlap_i$ do
                /* $OccSet_1$ is the first set of occurrences in $Overlap_i$ */
                /* Create the first element of each occurrence combination */
                $OverlapComb \leftarrow OverlapComb \cup \{\{occ\}\}$
            /* Extend each set of combinations in $OverlapComb$ */
            $j = 2$
            while $j \geq |Overlap_i|$ do
                $NewOverlapComb \leftarrow \emptyset$
                foreach $CombSet \in OverlapComb$ do
                    foreach $occ \in OccSet_j \in Overlap_i$ do
                        $NewCombSet \leftarrow CombSet \cup \{occ\}$
                        $NewOverlapComb \leftarrow NewOverlapComb \cup \{NewCombSet\}$
                $OverlapComb \leftarrow NewOverlapComb$
                $m = m + 1$
            foreach $CombSet \in OverlapComb$ do
                $OccFinal \leftarrow OccFinal \cup \{CombSet \cup NonOverlap\}$
Return $OccFinal$
```

$)\}, \{t_1 : (1,2,3)|t_1 : (7,8,10)|t_1 : (13,14,15)|t_2 : (1,2,3)|t_2 : (7,8,10)\}$ *is obtained. It can be observed that* s_1 *and* s_2 *are infrequent. However,* s_{union} *is frequent after the addition of the subgraphs.*

4.2.3 Extension of Patterns

Once size-1 patterns have been generated, the complete sequential pattern is generated by extending each size-1 pattern of each successive set of times. To do this, size-1 patterns are iteratively extended by checking the *mincom* and *minsup* constraints to connect other consecutive patterns to build a sequence of frequent subgraphs. This extension can be achieved by processing the times incrementally.

Figure 4.19 shows an incremental construction of a pattern beginning from $\{t_1, t_2\}$. This figure displays the parallel extensions of a pattern which occurs at t_1 and t_2. Let s, s' and s^* be frequent subgraphs extracted in graph additions. Additions between S_{t_1} and S_{t_2} result in a set of frequent subgraphs, such that $s = \langle \lambda, Occurrence(\lambda)\, in\, t_1, t_2 \rangle \in S_{t_1} + S_{t_2}$. Candidate extensions for these subgraphs can only be at t_2 and t_3 respectively (since gaps are not allowed). Now consider times t_2, t_3 and suppose that $s' = (\lambda', Occurrence(\lambda)'\, in\, t_2, t_3)$ is a frequent subgraph of $S_2 + S_3$. If s and s' have at least *minsup* occurrences verifying the temporal continuity constraint (*mincom*), then we can extend s with s' to obtain $P = \langle \{\lambda; \lambda'\}, \{t_1, t_2 : Occurrence(\lambda)\, in\, t_1, t_2; t_2, t_3 : Occurrence(\lambda')\, in\, t_2, t_3\} \rangle$. The pro-

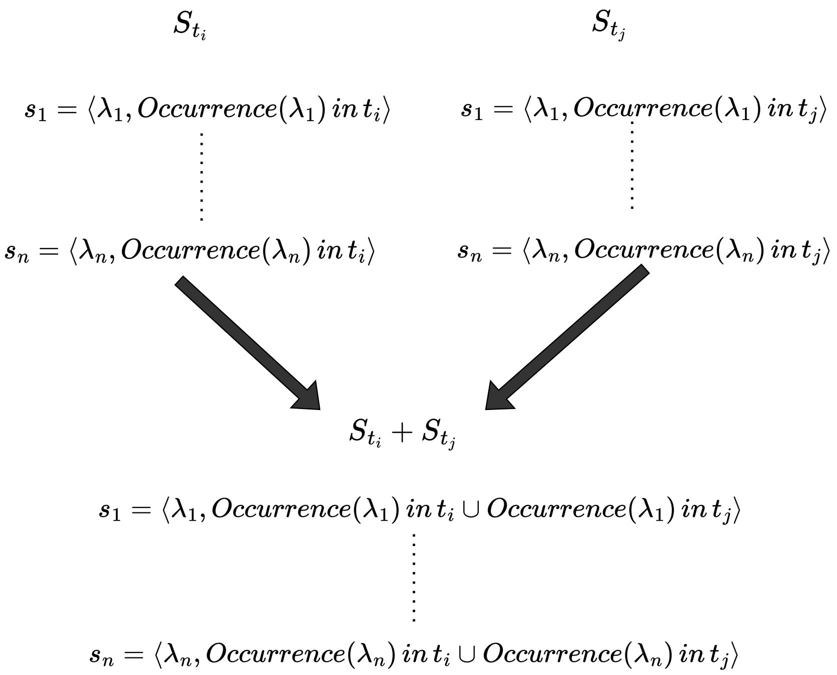

S_{t_i}
S_{t_j}
$s_1 = \langle \lambda_1, Occurrence(\lambda_1)\, in\, t_i \rangle$
$s_1 = \langle \lambda_1, Occurrence(\lambda_1)\, in\, t_j \rangle$
$s_n = \langle \lambda_n, Occurrence(\lambda_n)\, in\, t_i \rangle$
$s_n = \langle \lambda_n, Occurrence(\lambda_n)\, in\, t_j \rangle$
$S_{t_i} + S_{t_j}$
$s_1 = \langle \lambda_1, Occurrence(\lambda_1)\, in\, t_i \cup Occurrence(\lambda_1)\, in\, t_j \rangle$
$s_n = \langle \lambda_n, Occurrence(\lambda_n)\, in\, t_i \cup Occurrence(\lambda_n)\, in\, t_j \rangle$

Figure 4.18: Graph addition

Algorithm 9: *Union*: Compute the union of occurrences of subgraphs for a timestamp combination set

```
Input: a set of subgraphs: $S$, a timestamp combination set of size k: $T_i^k = \{t_i, ..., t_j\}$, minimum frequency of a
       pattern: $minsup$
Output: a set of size-1 patterns: $S_{union}$
$S_{union} \leftarrow \emptyset$
foreach $m = 1$ to $|S_t| \mid \forall S_t \in S, t \in T_i^k$ do
    /* $S_t$ is the set of subgraphs of $G_t$ */
    /* $|S_t|$ is the same for all $S_t \in S$ */
    $Ocurrence_{union} \leftarrow \cup_{\forall t \in T_i^k} Occurrence(\lambda_m)\, in\, t$ where $Occurrence(\lambda_m)\, in\, t \in S_t \in S$
    $s_{union} \leftarrow \langle \lambda_i, Occurrence_{union} \rangle$
    if $|Occurrence_{union}| \geq minsup$ then
        $S_{union} \leftarrow S_{union} \cup s_{union}$
Return $S_{union}$
```

cess continues until no more extensions can be performed. At each iteration, subgraphs can be used to extend patterns from the previous iteration, but they can also be starting points for new patterns.

For a sequence of 3 subgraphs, the pattern will be constructed and extended seven times for $T = \{t_1, t_2, t_3\}$: from $\{t_1\}$, from $\{t_2\}$, from $\{t_3\}$, from $\{t_1, t_2\}$, from $\{t_2, t_3\}$, from $\{t_1, t_3\}$, from $\{t_1, t_2, t_3\}$. Although the study of the combination $\{t_1, t_2\}$ does not bring more information compared to $\{t_1, t_2, t_3\}$, it allows discovering other patterns to be extended. All these time combinations are therefore necessary.

Example 16. *Consider the dynamic attributed graph in Fig. 4.15 and the constraints* $minvol = 2, maxvol = 4$ *and* $minsup = 2$. *We generate size-1 patterns from the addition of* G_{t_1} *and* G_{t_2}*:*

- $\langle (a_1 - a_2+, a_1 - a_2+, a_1 + a_2 \;=), \{t_1 : (1,2,3) | t_1 : (7,8,10) | t_1 : (13,14,15) | t_2 :$

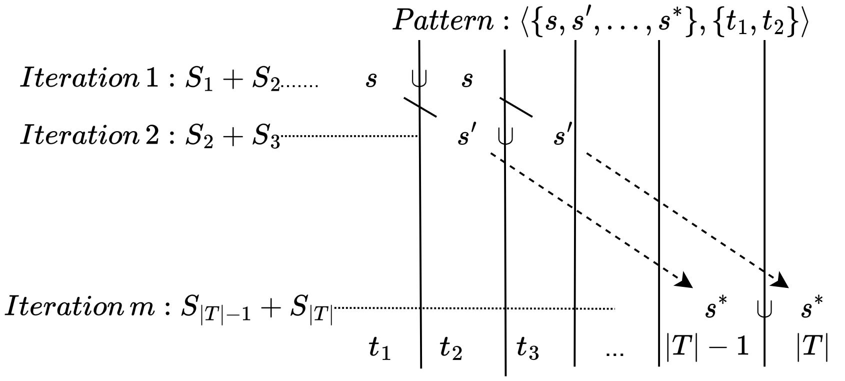

$Pattern : \langle\{s, s', \ldots, s^*\}, \{t_1, t_2\}\rangle$
$Iteration\,1 : S_1 + S_2$
s
$\uplus$
s
$Iteration\,2 : S_2 + S_3$
s'
$\uplus$
s'
$Iteration\,m : S_{|T|-1} + S_{|T|}$
s^*
$\uplus$
s^*
t_1
t_2
t_3
...
$|T|-1$
$|T|$

Figure 4.19: Additions and extensions of patterns from $\{t_1, t_2\}$

$(1,2,3)|t_2 : (7,8,10)\}\rangle$ *(in red)*

We generate size-1 patterns from the addition of G_{t_2} and G_{t_3}:

- $\langle\{(a1-a2+, a1+a2+, a1+a2-, a1-a2-)\}, \{t_2 : (1,4,5,6)|t_2 : (8,11,12,17)|t_2 : (13,14,15,16)|t_3 : (1,4,5,6)|t_3 : (8,11,12,17)\}\rangle$ *(in green)*
- $\langle\{(a1 = a2+, a1+a2+, a1+a2+, a1-a2-)\}, \{t_2 : (18,19,20,21)|t_3 : (18,19,20,21)\}\rangle$ *(in grey)*

We extend the size-1 patterns from the addition of G_{t_1} and G_{t_2} by verifying the temporal continuity constraint. Given a threshold $mincom = 1$, the pattern $\langle(a_1-a_2+, a_1-a_2+, a_1+a_2 =), \{t_1 : (1,2,3)|t_1 : (7,8,10)|t_1 : (13,14,15)|t_2 : (1,2,3)|t_2 : (7,8,10)\}\rangle$ is extended with the pattern $\langle\{(a1-a2+, a1+a2+, a1+a2-, a1-a2-)\}, \{t_2 : (1,4,5,6)|t_2 : (8,11,12,17)|t_2 : (13,14,15,16)|t_3 : (1,4,5,6)|t_3 : (8,11,12,17)\}\rangle$ as each of its occurrence has at least one common vertex with the candidate extension. Conversely, the pattern $\langle(a_1-a_2+, a_1-a_2+, a_1+a_2 =), \{t_1 : (1,2,3)|t_1 : (7,8,10)|t_1 : (13,14,15)|t_2 : (1,2,3)|t_2 : (7,8,10)\}\rangle$ is not extended by the pattern $\langle\{(a1 = a2+, a1+a2+, a1+a2+, a1-a2-)\}, \{t_2 : (18,19,20,21)|t_3 : (18,19,20,21)\}\rangle$ since it shares no common vertices in its occurrences with the candidate extension.

The Fig 4.20 shows the extracted solution beginning from $\{t_1, t_2\}$ and satisfying user defined constraints $minvol = 2, maxvol = 4, minsup = 4$ and $mincom = 1$. It illustrates a sequence of two subgraphs having respectively three and four vertices and at least one common vertex (for instance, the red vertex). Vertices are denoted as v_i as they are not dependent of occurrences. Edges are dotted to represent all connectivity possibilities between vertices.

4.3 Time Complexity of the Algorithm

In this section, we evaluate the *time complexity* of the FSSEMiner, i.e., the amount of time taken by an algorithm to run. To do so, we measure the time taken to execute each operation of the algorithm. This gives information about the variation in execution time of the algorithm when the number of operations varies in the algorithm. In particular, we highlight the *graph addition* strategy that allows to reduce the algorithm's time complexity.

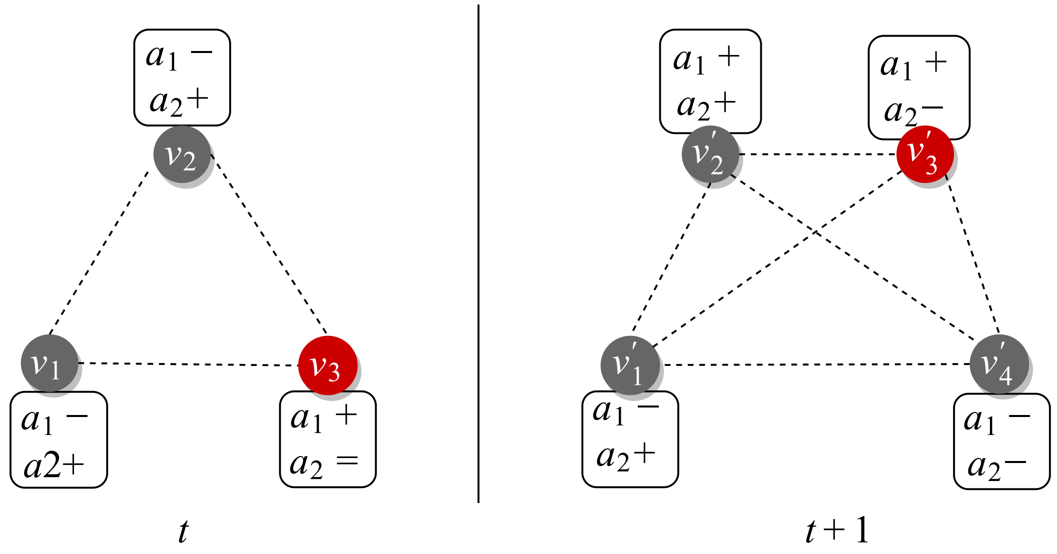


Figure 4.20: A FSSE solution beginning from $\{t_1, t_2\}$

4.3.1 Complexity of Subgraph Candidates Extraction

In the first step of the algorithm, the set of all possible attribute-value combinations of vertices, denoted $Cand_{\mathcal{A}\mathbb{D}}$, is generated. The time complexity of constructing $Cand_{\mathcal{A}\mathbb{D}}$ depends on the number of attributes and values. It is denoted as follows:

$$\begin{aligned} Com_{Cand_{\mathcal{A}\mathbb{D}}} &= \sum_{k=1}^{|\mathcal{A}|} \binom{|\mathcal{A}|}{k} |\mathbb{D}_{max}|^k \\ &= (|\mathbb{D}_{max}| + 1)^{|\mathcal{A}|}. \end{aligned}$$

where $\mathbb{D}_{max} = max \cup_{a \in \mathcal{A}} |\mathbb{D}_a|$ is the domain of an attribute with the maximum number of values.

Subgraphs candidates are then constructed based on the combinations of the elements of $Cand_{\mathcal{A}\mathbb{D}}$ and the chosen minimum and maximum volume of subgraphs. In the worst case, a subgraph candidate has a volume equal to the maximum number of vertices in $\mathcal{G}$, denoted $|V_{max}| = \cup_{t \in T} V_t$. So, the complexity of constructing all subgraph candidates is:

$$\begin{aligned} Com_{subgraphs\_generation} &= Com_{Cand_{\mathcal{A}\mathbb{D}}} + \sum_{vol=1}^{vol=|V_{max}|} |Cand_{\mathcal{A}\mathbb{D}}|^{vol} \\ &= (|\mathbb{D}_{max}| + 1)^{|\mathcal{A}|} + \frac{|Cand_{\mathcal{A}\mathbb{D}}|^{|V_{max}|+1} - 1}{|Cand_{\mathcal{A}\mathbb{D}}| - 1} - 1 \end{aligned}$$

Subgraph candidates are then extracted by traversing each graph G_t of the dynamic attributed graph $\mathcal{G}$. In the worst case, all vertices and edges of $\mathcal{G}$ are traversed. So the time complexity of traversing $\mathcal{G}$ is:

$$Com_{graph\_traversal} = |T| \times (|V_{max}| + |E_{max}|)$$

where $|T|$ is the number of graphs to be traversed, $V_{max} = \cup_{t \in T} V_t$ and $E_{max} = \cup_{t \in T} E_t$.

To conclude, the time complexity of extracting subgraph candidates in $\mathcal{G}$ is:

$$\begin{aligned} Com_{subgraph\_extraction} &= Com_{subgraphs\_generation} + Com_{graph\_traversal} \\ &= (|\mathbb{D}_{max}| + 1)^{|\mathcal{A}|} + \frac{|Cand_{\mathcal{A}\mathbb{D}}|^{|V_{max}|+1} - 1}{|Cand_{\mathcal{A}\mathbb{D}}| - 1} - 1 \\ &+ |T| \times (|V_{max}| + |E_{max}|) \end{aligned}$$

4.3.2 Complexity of Graph Addition

In the second step, the graph addition is performed to generate size-1 patterns using the subgraphs generated from the previous step. Consider the complexity of the addition of k graphs. It requires performing additions of two graphs $(k-1)$ times. Graph addition is applied to all sets of timestamp combinations, denoted as T_i^k, $1 \leq k \leq |T|$. So for every timestamp combination $T_i^k \subseteq T$, $(k-1)$ additions have to be performed. The total number of timestamp combinations is equal to $\sum_{k=1}^{|T|} \binom{|T|}{k}$. So the time complexity of the generation of size-1 patterns by graph addition is equal to:

$$\begin{aligned} Com_{addition} &= \sum_{k=1}^{|T|} \binom{|T|}{k} (k-1) \\ &= \sum_{k=1}^{|T|} k \times \binom{|T|}{k} - \sum_{k=1}^{|T|} \binom{|T|}{k} \\ &= (|T| \times 2^{|T|-1}) - (2^{|T|} - 1) \\ &= 2^{|T|-1} \times (|T| - 2) + 1 \end{aligned}$$

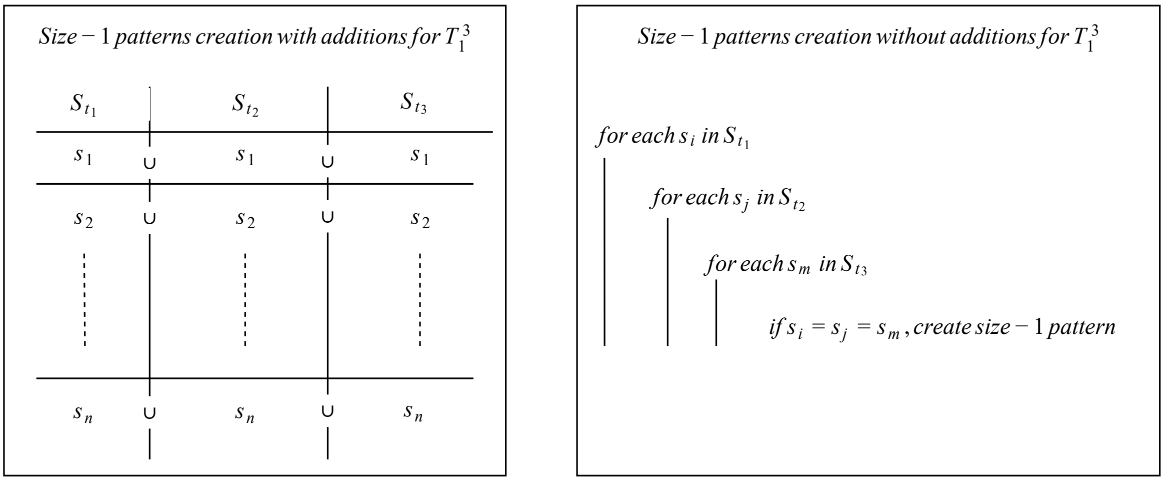
As we can see in Fig. 4.21 b), without graph addition, we would perform a huge amount of subgraph traversals for the generation of patterns of size-1. In the worst case, the number of subgraphs extracted in a timestamp is $|V_{max}| \cup_{t \in T} V_t$. So, the time complexity of the generation of size-1 patterns without the addition strategy is:

$$\begin{aligned} Com_{noaddition} &= \sum_{k=1}^{|T|} \binom{|T|}{k} \times |V_{max}|^k \\ &= (|V_{max}| + 1)^{|T|} \end{aligned}$$

The complexity of size-1 patterns generation with addition only depends on the number of timestamps $|T|$ while without addition, it depends on both the timestamps and the number of vertices. In a nusthell, $Com_{noaddition} > Com_{addition}$.

4.3.3 Complexity of Extension

In the third step, size-1 patterns are extended for every timestamp combination $T_i^k \subseteq T$. As mentioned above, in the worst case, the maximal number of subgraphs extracted in a timestamp is $|V_{max}|$. Consequently, for every timestamp combination $T_i^k \subseteq T$, the maximal number of patterns that can be generated by the successive extensions is $|V_{max}|^{|T|-1}$.


Size − 1 patterns creation with additions for T_1^3
S_{t_1}
S_{t_2}
S_{t_3}
s_1 ∪ s_1 ∪ s_1
s_2 ∪ s_2 ∪ s_2
s_n ∪ s_n ∪ s_n
Size − 1 patterns creation without additions for T_1^3
for each s_i in S_{t_1}
for each s_j in S_{t_2}
for each s_m in S_{t_3}
if $s_i = s_j = s_m$, create size − 1 pattern

a)

b)

Figure 4.21: Addition for the timestamp combination T_1^3

As the total number of timestamp combinations is $\sum_{k=1}^{|T|}\binom{|T|}{k}$, the time complexity of generating all extensions is thus equal to:

$$\begin{aligned} Com_{extension} &= \sum_{k=1}^{|T|}\binom{|T|}{k} \times |V_{max}|^{|T|-1} \\ &= (2^{|T|}-1) \times |V_{max}|^{|T|-1} \end{aligned}$$

4.3.4 Total Complexity

Based on the time complexity of subgraphs extraction, addition and extension, the complexity of FSSEMiner algorithm is :

$$\begin{aligned} Com_{FSSEMiner} &= Com_{subgraph\_extraction} + Com_{addition} + Com_{extension} \\ &= (|\mathbb{D}_{max}|+1)^{|\mathcal{A}|} + \frac{|Cand_{\mathcal{A}\mathbb{D}}|^{|V_{max}|+1}-1}{|Cand_{\mathcal{A}\mathbb{D}}|-1} - 1 + |T| \times (|V_{max}| + |E_{max}|) \\ &+ 2^{|T|-1} \times (|T|-2) + 1 \\ &+ (2^{|T|}-1) \times |V_{max}|^{|T|-1} \end{aligned}$$

5 Experimental Assessments of FSSEMiner

This section reports the experiments we carry out to validate the performance of the proposed FSSEMiner algorithm. The objectives of the experiments are: (i) to evaluate the scalability of the algorithm with regard to different graph characteristics, and (ii) to analyse patterns found by the proposed algorithm to assess whether they are useful in different application domains. So, first, we present in this section the conditions of the experiments (Section 5.1). Second, we present the quantitative evaluation of the algorithm to meet the objective (i) (Section 5.2). Finally, we present the qualitative evaluation of the algorithm to meet the objective (ii)(Section 5.3). The process of our experiments is illustrated in Fig. 4.22.

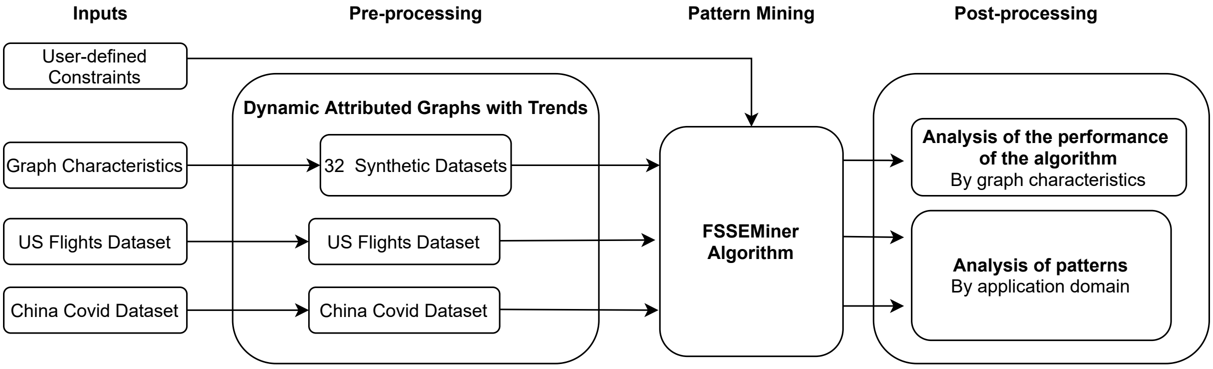


Figure 4.22: Experimental process overview

5.1 Experimental conditions

5.1.1 Technical Environment

The algorithm was implemented in C ++. The pre-processing of datasets were done in the programming language Python 3.8. Experiments were conducted on 16 CPUs x Intel(R) Xeon(R) Silver 4216 CPU @ 2.10GHz. A virtual machine is installed on this hardware with 64 GB in terms of RAM and 250GB in terms of disk size. The technical details of our experiments are available on the website `https://gitlab.com/2573869/patternmining`.

5.1.2 Synthetic Datasets

For the quantitative evaluation, 32 synthetic dynamic attributed graphs were generated according to different graph characteristics: the size of the sequence (i.e., the number of timestamps in the dynamic attributed graph), the number of vertices and edges per timestamp and the number of vertex attributes. Algorithm 10 gives the details of the generation of a synthetic dataset. First, we create the graphs of the dynamic attributed graph according to a given size of the sequence $|T|$ (*lines 1-3*). For each graph, we create a given number of vertices $|V|$ (*lines 4-6*). Then, we associate each vertex with a given number of attributes $|A|$, for which values follow a uniform distribution (*line 7-8*). Moreover, for each graph, we also create a given number of pairs of vertices (or edges) $|E|$ based on a uniform distribution (*lines 9-11*).

5.1.3 Real-world Datasets

To run our qualitative evaluation, 2 real-world datasets are used (Table 4.4): (i) a dataset about US Domestic Flights derived from the RITA[1] database and (ii) two datasets about the COVID-19 cases and travel flows in China derived from the Harvard Dataverse database. The first objective is to assess whether the algorithm discovers interesting patterns to study the consequences of two disruptive events in the real-world: respectively the impact of the Katrina hurricane in 2005 on US Domestic Flights and the impact of the COVID on travel flows in China in 2020 and 2022. The second objective is to assess the advantage of our new pattern compared to existing ones on real-world applications. For this purpose, the Domestic US Flights dataset is as it is used in other research works, contrary to the China Covid Dataset.

[1] Research and Innovative Technology Administration

Algorithm 10: *GenerateSyntheticDataset*: Function to generate a synthetic dataset

```
Input: the number of vertices per graph: $|V|$, the number of edges per graph: $|E|$, the number of attributes per
       vertex: $|A|$, a set of value domains : $\mathbb{D} = \{\mathbb{D}_1, ...\mathbb{D}_k, ...\mathbb{D}_{|\mathcal{A}|}\}$, the number of timestamps: $|T|$
Output: A dynamic attributed graph: $\mathcal{G}$
for $i = 1$ to $|T|$ do
    $V_{t_i} \leftarrow \emptyset, E_{t_i} \leftarrow \emptyset$
    Create $G_{t_i} = (V_{t_i}, E_{t_i}, \lambda_{t_i})$
    for $j = 1$ to $|V|$ do
        Create $v_j$
        $V_{t_i} \leftarrow V_{t_i} \cup \{v_j\}$
        for $k = 1$ to $|A|$ do
            Create $a_k = UniformDistribution(min(\mathbb{D}_k), max(\mathbb{D}_k))$ where $a_k \in \lambda_{t_i}(v_j)$
    for $l = 1$ to $|E|$ do
        $e_l = (v_m, v'_m)$ where $v_m, v'_m \in V_{t_i}$, $m = UniformDistribution(1, |V|)$ and
          $m' = UniformDistribution(1, |V|)$
        $E_{t_i} \leftarrow E_{t_i} \cup \{e_l\}$
Return $\mathcal{G}$
```

US Flights Dataset. The RITA dataset[2] contains US domestic flights by major air carriers. From this dataset, we construct a dynamic attributed graph. The latter aggregates US domestic flights data over each week during the Katrina hurricane period from 01/08/2005 to 25/09/2005. In other words, the gap between each timestamp t and $t+1$ is a week and there are 8 timestamps in the dynamic attributed graph. Graph vertices represent US airports and are connected by an edge if there is at least a flight connecting them during the time period. There are 280 vertices and, 1206 edges per timestamp in average. We consider 4 vertex attributes for which the values change over time: (i) *the number of departure delays*, (ii) *the number of arrival delays*, (iii)*the number of cancelled flights* and (iv) *the number of diverted flights* (whose destination airport has been diverted).

China Covid Dataset. The Harvard Dataverse database proposes a dataset about the COVID-19 daily cases in Chinese cities[3] since 2020 and a dataset about travel flows based on Baidu Mobility Index[4] between Chinese cities. From these two datasets, we generate two dynamic attributed graphs over two periods of time during which peaks in the number of daily COVID cases were observed: (i) from 15/02/2020 to 04/03/2020 and (ii) from 17/04/2022 to 05/05/2022. For both dynamic attributed graphs, data are aggregated every 3 days. In other words, the gap between each timestamp t and $t+1$ is 3 days and there are 6 timestamps in each dynamic attributed graph. Vertices represent Chinese cities and are connected with an edge if the mobility index between cities is not null (i.e., if there are travel flows between cities). We consider 4 vertex attributes: (i) the *city size* with a value equal to « small », « medium », « big » or « mega » city according to the population[5], *the total number of new COVID cases*, (iii) *the total number of new deaths* and (iv) *the total number of new recoveries* since the beginning of the two periods. The value of the size of the city is fixed (i.e., does not change over time) while the value of the three other attributes changes over time. There are 232 vertices and, 13260 edges

[2] `https://www.transtats.bts.gov/`

[3] `https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/MR5IJN`

[4] `https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/FAEZIO&version=21.1`, `https://qianxi.baidu.com/`

[5] small city : less than 500.000 residents, medium city: between 500.000 and 1 million residents, big city: between 1 million and 5 million residents, mega city: more than 5 million residents

Table 4.4: Real datasets description

Dynamic Attributed Graph	Number of Vertices in average per timestamp	Number of Edges in average per timestamp	Number of Attributes	Number of Timestamps
US Flights Dataset	280	1206	4	8
China Covid Dataset	232	13260	4	6

per timestamp in average.

Transformation into trends. As discussed in the Section 3.1, it is desirable to transform dynamic attributed graphs into trend graphs in the case we have numerical attribute values. To construct the trend graphs, we apply the transformation rules in the Section 3.1. More precisely, two different discretization strategies of attribute values were used for each dataset.

For the US Flights dataset, the value of all attributes (*the number of departure delays, the number of arrival delays, the number of cancelled flights and the number of diverted flights*) change over time. A simple discretization is applied to the value of all attributes to transform them into trends. The trend of each attribute a of vertex v at a timestamp t is computed according to the following rule:

$$trend(v,a,t) = \begin{cases} +, & \text{if } value(v,a,t) < value(v,a,t+1) \\ =, & \text{if } value(v,a,t) = value(v,a,t+1,) \\ -, & \text{if } value(v,a,t) > value(v,a,t+1) \\ \emptyset, & \text{otherwise} \end{cases}$$

For the China Covid dataset, the values of the attribute *city size* are not transformed into trends, as they are fixed over time. Conversely, the values of the attributes *the total number of new COVID cases*, *the total number of new deaths* and *the total number of new recoveries* change over time. More precisely, they increase over time as they display the accumulated number of new COVID cases, deaths and recoveries since the beginning of the period. Therefore, to transform them into trends, a discretization in terms of the proportion of the increase is applied as follows:

$$trend(v,a,t) = \begin{cases} =, & \text{if no new cases between } t \text{ and } t+1 \\ +, & \text{if } value(v,a,t+1) - value(v,a,t) \in]0,5] \\ ++, & \text{if } value(v,a,t+1) - value(v,a,t) \in]5,15] \\ +++, & \text{if } value(v,a,t+1) - value(v,a,t) \in]15,] \end{cases}$$

5.1.4 Choice of constraints

As presented in Section 4.15, the choice of the value of constraints $minvol, maxvol, mincom$ and $minsup$ meets two objectives: (i) users' needs and objectives (What are the questions they are trying to answer ?) and (ii) the need to reduce the search space according to the dataset characteristics. Generally, the value of constraints are adjusted iteratively to meet the two objectives. In other terms, the mining algorithm is run iteratively according to different values of constraints to find the ones that meet the two objectives.

In the case (i) users want to capture the changes in the structure of entity groups over time, the value of the *mincom* constraint needs to be significantly lower than *maxvol* (by default it is set to 1) to extract patterns including the addition and removal of new entities in groups over time. Otherwise, they will set a value *mincom* close to *maxvol*.

5.2 Quantitative Evaluation

We conduct a quantitative evaluation of the performance of the FSSEMiner algorithm based on the synthetic datasets. Since a new pattern has been proposed, as far as we know, there exist no algorithms that can be compared with the proposed algorithm. So the performance of the algorithm is evaluated by comparing its runtime and scalability on different graph characteristics: the number of timestamps (Section 5.2.1), the number of attributes (Section 5.2.2) and the number of vertices and edges (Section 5.2.3). The user-defined constraints are set as follows: $minvol = maxvol = 2$, $mincom = 1$, $minsup = 60\%$ of the average number of vertices per graph in the dynamic attributed graph. Indeed, we set the value of *minsup* according to the number of vertices per graph since we observe that the number of occurrences of patterns depends on the number of vertices.

5.2.1 Impact of the number of timestamps

The first experiment assesses the impact of the number of timestamps on the algorithm's performance. To do so, the algorithm was run on different dynamic attributed graphs having a different number of timestamps. The other graph characteristics were fixed: 2000 vertices and 8000 edges per graph and 2 attributes. Fig.4.23 shows that when the number of timestamps increases, both the algorithm's execution times and the number of produced patterns increase exponentially. More precisely, the algorithm's execution times increase slowly until 14 timestamps ($\leq 874s$) and increase very quickly beyond this threshold. Indeed, the number of additions and extensions that must be performed is even more important when the number of timestamps is high (see Section 4.3).

5.2.2 Impact of the number of attributes

The second experiment studies the impact of the number of attributes on the algorithm's performance. To do so, the algorithm was run on different dynamic attributed graphs having a different number of attributes. The other graph characteristics constraints were fixed: 2000 vertices and 8000 edges per graph and 8 timestamps. Fig.4.24 shows that the number of produced patterns increases linearly with the number of attributes. Indeed, the increase of the number of attributes increases the number of generated candidate subgraphs that have to be extracted in the dynamic attributed graph. Despite the search space growth, the runtimes of the algorithm remain low for less than 8 attributes ($< 672s$). This is notably thanks to the graph addition strategy that allows reducing the runtimes consumed by avoiding exponential subgraphs traversal during the generation of candidate subgraphs (see Section 4.3). In a nutshell, the impact of the increase of the attributes number on the overall performance of the algorithm is relatively small.

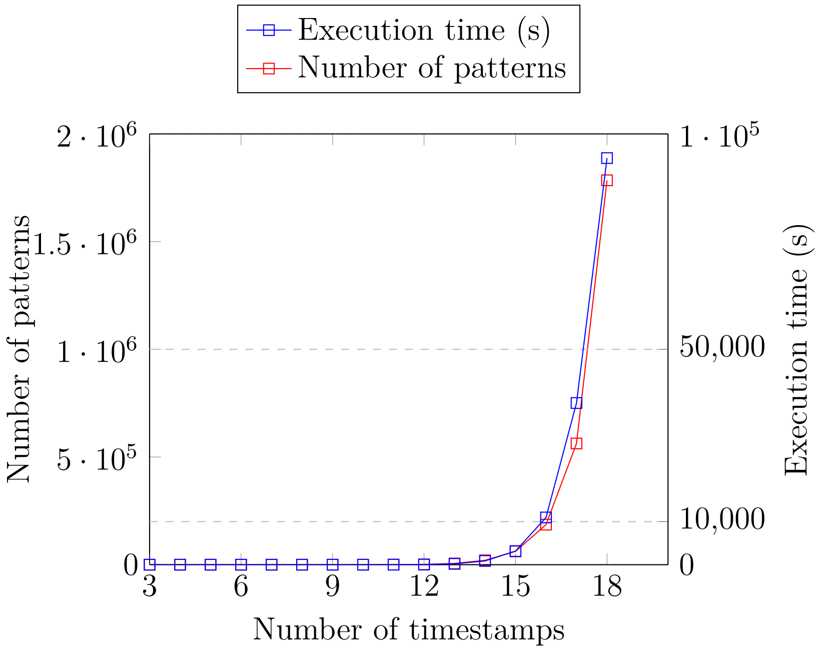


Figure 4.23: Impact of the number of timestamps (synthetic datasets)

5.2.3 Impact of the number of vertices and edges

The third experiment evaluates the impact of the number of vertices and edges per graph (i.e, per G_t) in the dynamic attributed graph on the algorithm's performance. To do so, the algorithm was run on different dynamic attributed graphs having a different number of vertices and edges per graph. The other graph characteristics were fixed: 2 attributes and 8 timestamps. Fig.4.25 shows that when the number of vertices and edges per graph increases, the algorithm's runtimes grow quickly while the number of produced patterns is relatively more stable. Increasing the number of vertices and number of edges per graph implies that more vertices and edges are traversed to find occurrences in the dynamic attributed graph (see Section 4.3).

5.3 Qualitative Evaluation

5.3.1 Analysis of US Flights patterns

We carry out a qualitative analysis of patterns extracted by the FSSEMiner algorithm in the US Flights dataset. The objective is to find interesting patterns that show the impact of the Katrina hurricane on the US Flights traffic. To do so, the user-defined constraints are set as follows:

- $minvol = 2$ and $maxvol = 4$ because we want to extract patterns representing the evolution of relatively small airport networks;
- $mincom = 1$ because we want to extract patterns capturing the impact of the disruptive event on the airport network structure (addition and removal of new airport);
- $minsup = 25$ meaning that patterns should be frequent at least 25 times in time and space. We observed through several executions of the algorithm that frequent

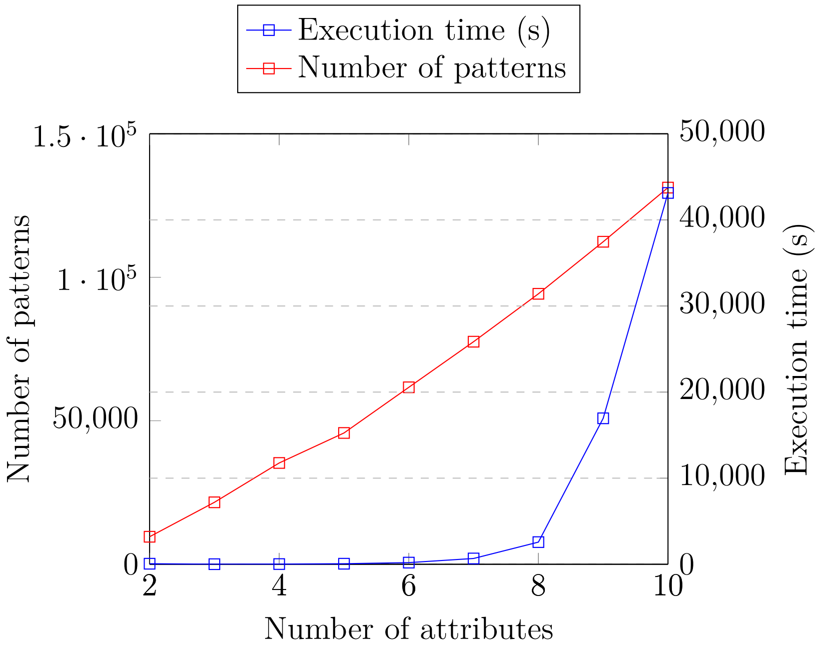


Figure 4.24: Impact of the number of attributes (synthetic datasets)

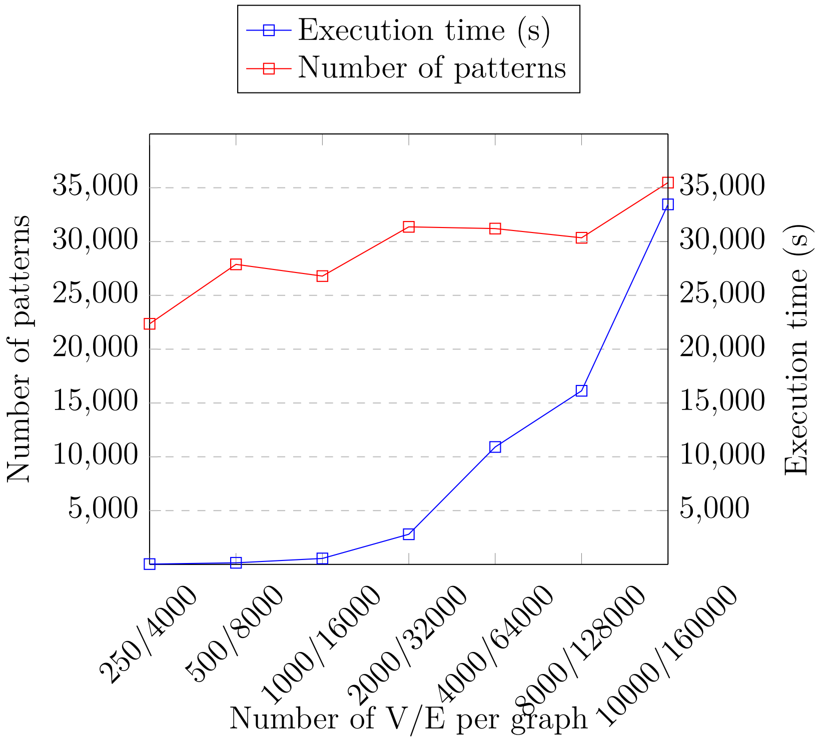
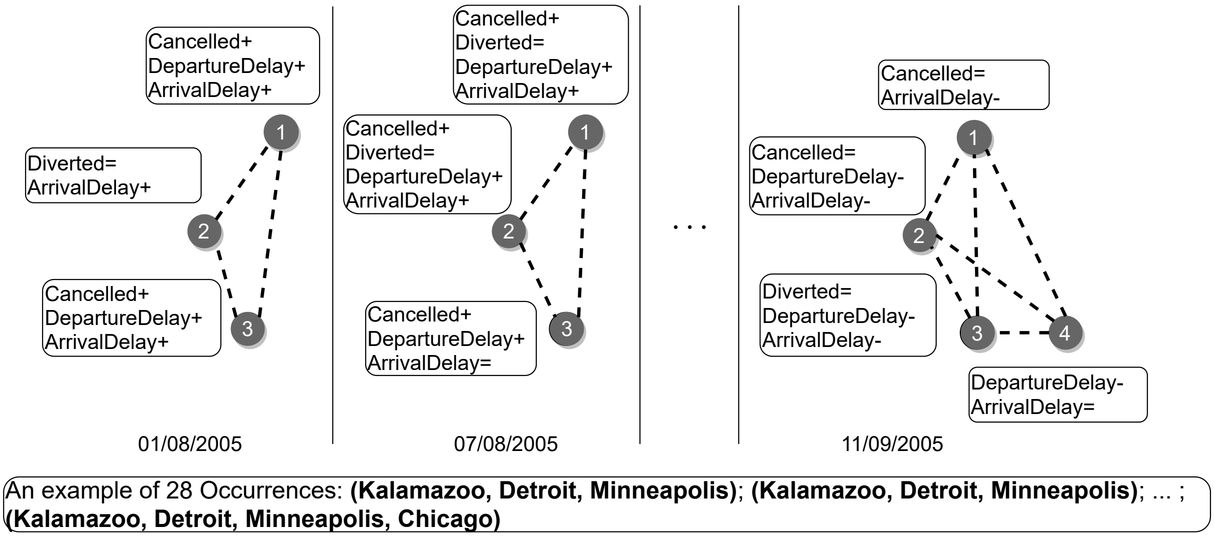


Figure 4.25: Impact of the number of vertices and edges (synthetic datasets)

patterns are extracted from this value of *minsup*.

Fig. 4.26 shows an example of a FSSE pattern extracted from the US Flights traffic dataset. Using the formal notation, this pattern is as follows:

$$(Cancelled + DepartureDelay + ArrivalDelay+, Diverted - ArrivalDelay+,\\ Cancelled + DepartureDelay + ArrivalDelay+);$$



An example of 28 Occurrences: **(Kalamazoo, Detroit, Minneapolis)**; **(Kalamazoo, Detroit, Minneapolis)**; ... ; **(Kalamazoo, Detroit, Minneapolis, Chicago)**

Figure 4.26: A FSSE pattern extracted from the US Flights dataset

$$(Cancelled+Diverted-DepartureDelay+ArrivalDelay+, Cancelled+Diverted-DepartureDelay+ArrivalDelay+, Cancelled+DepartureDelay+ArrivalDelay=);$$
$$...;$$
$$(Cancelled=ArrivalDelay-, Cancelled=DepartureDelay-ArrivalDelay-, Diverted-DepartureDelay-ArrivalDelay-, DepartureDelay-ArrivalDelay=)$$

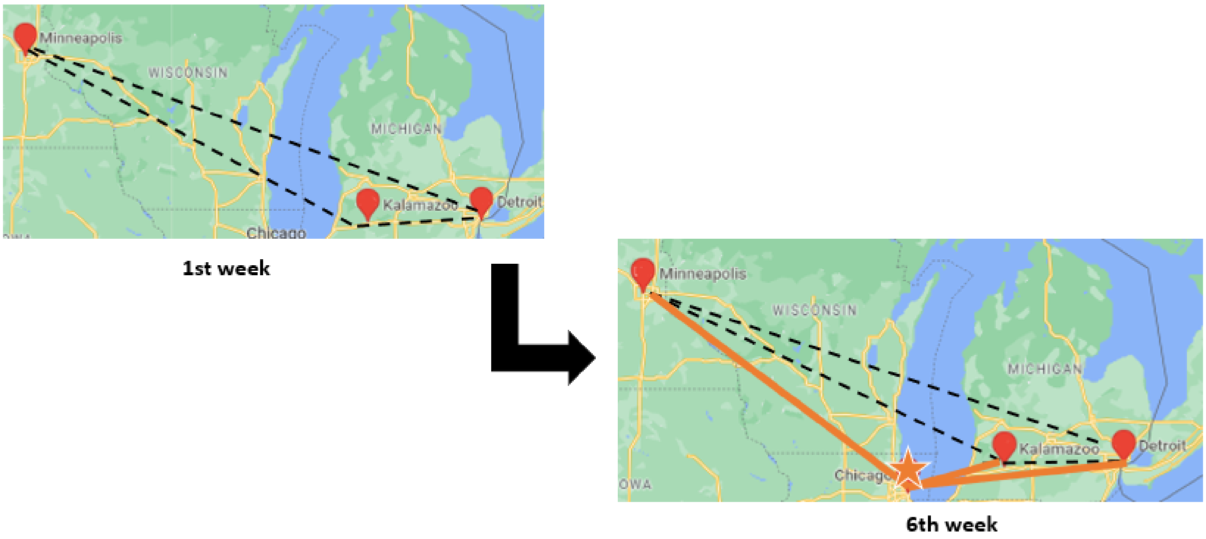
This pattern depicts a frequent evolution of the departure/arrival delays and cancelled/diverted flights in several aiport networks composed by 3 to 4 airports (respecting *minvol* and *maxvol* constraints). This is a sequence of size 6, i.e., an evolution over 6 timestamps (here, weeks). For clarity of presentation, only the first, the second and the last airport networks of the sequence are shown. Vertices are airports and edges are the possible flight connections between airports. Vertex attributes are the number of departure delays, the number of arrival delays, the number of cancelled flights and the number of diverted flights. In the following, several observations on the pattern are presented.

First, the FSSE pattern appears 28 times in time and space in the dataset (respecting the *minsup* constraint). In other words, 28 airport networks were affected by the hurricane in the same manner. For instance, the following airport network has the pattern:

$$(Kalamazoo, Detroit, Minneapolis);$$
$$(Kalamazoo, Detroit, Minneapolis);$$
$$...;$$
$$(Kalamazoo, Detroit, Minneapolis, Chicago).$$

Second, it is observed that in the first two weeks, when the hurricane came, delays and cancellations increased at destination and arrival airports, while diverted flights always remained the same. It shows that airlines preferred to cancel and delay flights rather than divert flights when the hurricane came. It may be because diverting flights generates higher costs than delays and cancellations for airlines. Six weeks later, it is noticed that cancellations and diverted flights became stable while delays decreased because the hurricane became weaker.

Third, the pattern shows an evolution in terms of topology. The network of three



1st week

6th week

Figure 4.27: Map of US to show the change in the airport network

airports became four. This means that new flight routes via a new airport were added by airlines in the last week 11/09/2005. For instance, in the previously mentionned occurrence of the pattern, the new airport, *Chicago*, was added to the airport network $(Kalamazoo, Detroit, Minneapolis)$. Referring to the Fig 4.27, we notice that the new added airport, *Chicago*, is located at the center position of the previous airport network $(Kalamazoo, Detroit, Minneapolis)$, which ensures that airline connections are more convenient as it is close to all other airports.

Finally, we compare the extracted FSSE pattern to the closest pattern, the recurrent patterns (Cheng et al., 2017), to evaluate the advantage of the FSSE pattern. To do so, we study the reccurrent patterns extracted by the RPMiner algorithm proposed in (Cheng et al., 2017) from the same US Flights dataset. The following pattern is extracted:

$$\begin{gathered}(\langle(Bangor : DepartureDelay + | Boston : DelayArrival + | \\ NewportNews : DelayDeparture+) \\ (Augusta : Cancelled - | Bangor : \\ Cancelled-)\rangle, \{01/08/2005, 08/08/2005, 29/08/2005, 05/09/2005\})\end{gathered}$$

This recurrent pattern is also illustrated in Fig 4.28. It depicts an evolution of the departure/arrival delays and cancelled/diverted flights in an airport network $(Bangor, Boston, NewportNews)$ over 2 weeks (since it is a sequence of size 2). Moreover, it is frequent 4 times in the dataset (at weeks 01/08/2005, 08/08/2005, 29/08/2005 and 05/09/2005). Contrary to the previous FSSE pattern, this reccurrent pattern represents an evolution of one specific aiport network. Thus, the advantage of the FSSE pattern is to be representative of the evolutions of several airport networks. This is possible because the FSSE pattern takes into account the spatial dimension, i.e., that an evolution can occur at different locations.

5.3.2 Analysis of China Covid patterns

We carry out a qualitative analysis of patterns extracted by the FSSEMiner algorithm in the China Covid Dataset. The goal is to find interesting patterns to analyze the

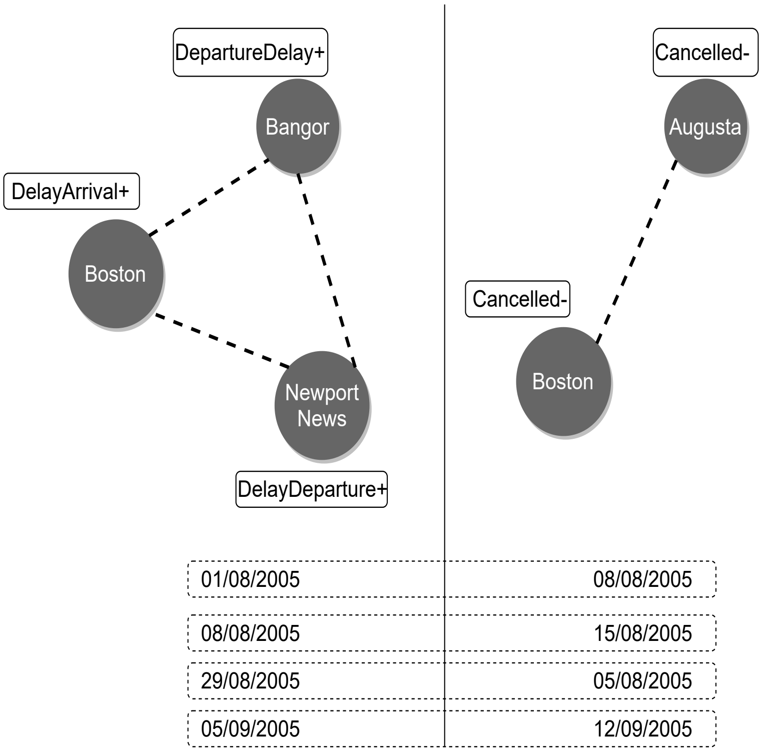


Figure 4.28: A reccurrent pattern extracted from the US Flights dataset in (Cheng et al., 2017)

transmission of COVID in 2020 and its variant Omicron in 2022 through travel flows between cities in China. To do so, the user-defined constraints are set as follows:

- *minvol* = 5 and *maxvol* = 8 because we want to extract patterns describing the evolution of COVID cases in networks of cities having diverse sizes (small, medium, big, mega) ;
- *mincom* = 5 because we are not interested in the addition and removal of new cities in the city networks ;
- *minsup* = 15 meaning that patterns should be frequent at least 15 times in time and space. We observed through several executions of the algorithm that frequent patterns are extracted from this value of *minsup*.

Two FSSE patterns are extracted by the algorithm. Fig 4.29 a) illustrates one extracted pattern to analyse the transmission of COVID in a period during which COVID cases were booming (from 15/02/2020 to 04/03/2020). Similarly, Fig 4.29 b) illustrates another extracted pattern to analyse the transmission of the Omicron variant from 17/04/2022 to 05/05/2022. Here, vertices are cities and edges are the possible travel flows between cites. Vertex attributes are the city sizes (small, medium, big and mega), the total number of new COVID cases, new recoveries and new deaths. The two FSSE patterns depict frequent evolutions of the total number of new COVID cases/new recoveries/new deaths in several city networks composed by 5 to 8 cities (respecting *minvol* and *maxvol* constraints). They are sequences of size 7 representing evolutions over 7 timestamps (here, 7 days). The first pattern occurs 21 times in the dataset, while the second occurs 14 times. For clarity of presentation, only the first, the second and the last timestamps of the sequence

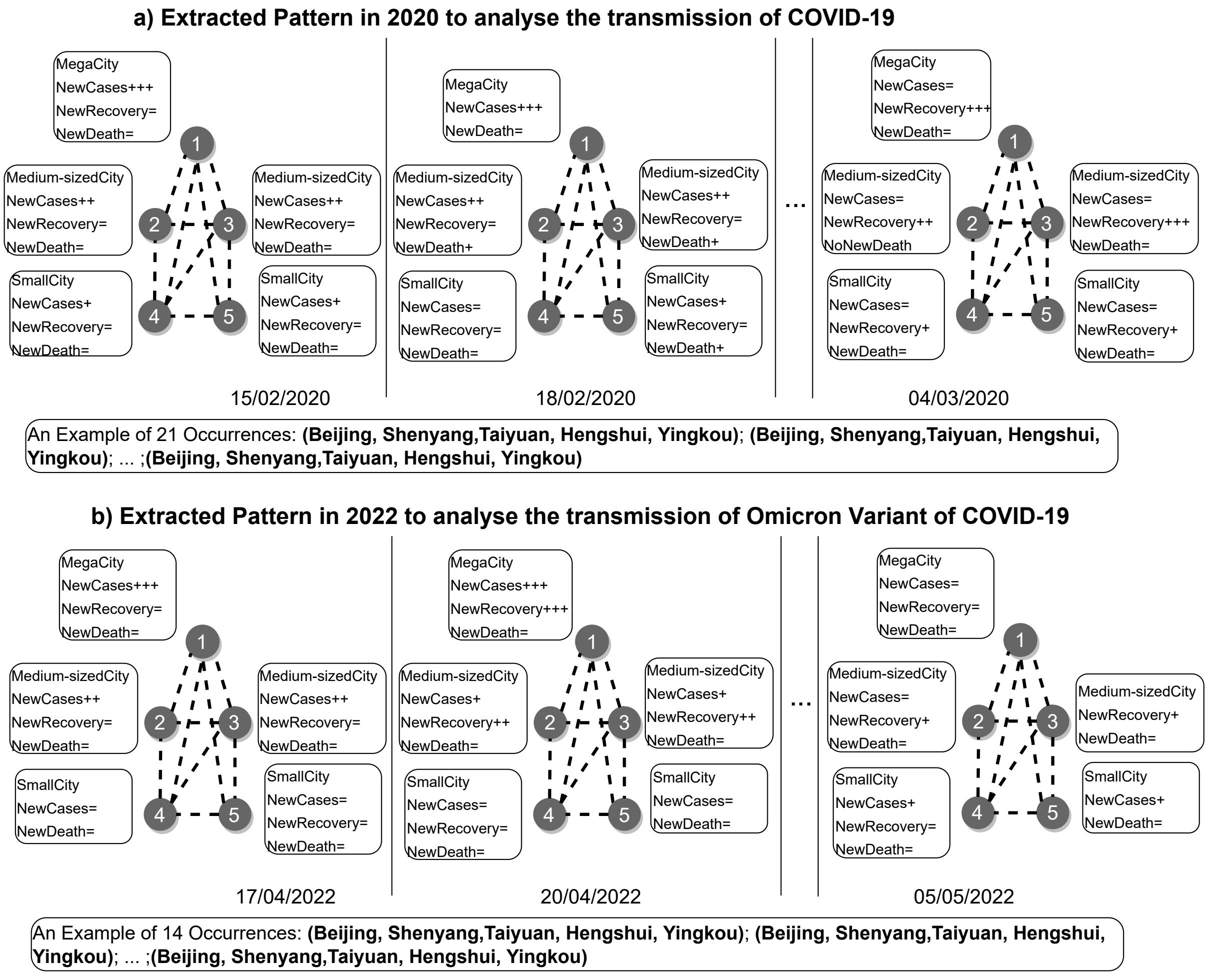

a) Extracted Pattern in 2020 to analyse the transmission of COVID-19
MegaCity NewCases+++ NewRecovery= NewDeath=
Medium-sizedCity NewCases++ NewRecovery= NewDeath=
Medium-sizedCity NewCases++ NewRecovery= NewDeath=
SmallCity NewCases+ NewRecovery= NewDeath=
SmallCity NewCases+ NewRecovery= NewDeath=
1
2
3
4
5
15/02/2020
MegaCity NewCases+++ NewDeath=
Medium-sizedCity NewCases++ NewRecovery= NewDeath+
Medium-sizedCity NewCases++ NewRecovery= NewDeath+
SmallCity NewCases= NewRecovery= NewDeath=
SmallCity NewCases+ NewRecovery= NewDeath+
18/02/2020
...
MegaCity NewCases= NewRecovery+++ NewDeath=
Medium-sizedCity NewCases= NewRecovery++ NoNewDeath
Medium-sizedCity NewCases= NewRecovery+++ NewDeath=
SmallCity NewCases= NewRecovery+ NewDeath=
SmallCity NewCases= NewRecovery+ NewDeath=
04/03/2020
An Example of 21 Occurrences: (Beijing, Shenyang,Taiyuan, Hengshui, Yingkou); (Beijing, Shenyang,Taiyuan, Hengshui, Yingkou); ... ;(Beijing, Shenyang,Taiyuan, Hengshui, Yingkou)
b) Extracted Pattern in 2022 to analyse the transmission of Omicron Variant of COVID-19
MegaCity NewCases+++ NewRecovery= NewDeath=
Medium-sizedCity NewCases++ NewRecovery= NewDeath=
Medium-sizedCity NewCases++ NewRecovery= NewDeath=
SmallCity NewCases= NewDeath=
SmallCity NewCases= NewRecovery= NewDeath=
17/04/2022
MegaCity NewCases+++ NewRecovery+++ NewDeath=
Medium-sizedCity NewCases+ NewRecovery++ NewDeath=
Medium-sizedCity NewCases+ NewRecovery++ NewDeath=
SmallCity NewCases= NewRecovery= NewDeath=
SmallCity NewCases= NewDeath=
20/04/2022
...
MegaCity NewCases= NewRecovery= NewDeath=
Medium-sizedCity NewCases= NewRecovery+ NewDeath=
Medium-sizedCity NewRecovery+ NewDeath=
SmallCity NewCases+ NewRecovery= NewDeath=
SmallCity NewCases+ NewDeath=
05/05/2022
An Example of 14 Occurrences: (Beijing, Shenyang,Taiyuan, Hengshui, Yingkou); (Beijing, Shenyang,Taiyuan, Hengshui, Yingkou); ... ;(Beijing, Shenyang,Taiyuan, Hengshui, Yingkou)

Figure 4.29: Two FSSE patterns extracted from COVID dataset

are shown. In the following, several observations on the pattern are presented.

First, the two chosen patterns highlight the transmission of COVID in a mixed city network which is composed of different city sizes. We notice that the city sizes and the new case numbers are strongly correlated. In 2020 and 2022, COVID spreads very quickly in medium and megacities, while new cases in small cities have shown almost zero growth. It is probably because in small cities, the transport connections are much easier to control. For example, a small city has in general only two train stations and one airport while in medium-sized and big cities, it could have up to 109 train stations and 12 airports. Moreover, the flow is in general 30 times higher than small cities, which makes it much more difficult to miss any positive case.

Second, it is observed that the COVID caused many severe consequences for the period from 15/02/2020 to 04/03/2020. The number of deaths began to increase in three days after the emergence of new COVID cases (Fig 4.29 a)). Moreover, it took in average more than 10 days for recovery. While in 2022, the virulence of variant Omicron was much weaker. From 17/04/2022 to 05/05/2022, there were almost no deaths. Moreover, the recoveries began to emerge only three days after new detected cases (Fig 4.29 b)).

To conclude, these two patterns allow studying virus transmission in different scales of cities (among big cities, small cities, medium-sized or mixed city network). We are able to compare the virulence of different virus variants, which provides insights about the relative efficient measures to adopt (e.g. quarantine, isolation policies). In future works, we plan to generate edges to model the flows of different transportation modes (air flights, trains and bus instead of the total flow) to analyse their individual impact on virus transmission.

6 Conclusion

Querying *Temporal Graphs* (TG) allows finding isolated pieces of information to answer straightforward business questions concerning temporal interconnected data ('*What*', '*Who*', '*Where*' and '*When*'). By combining information pieces from TG, we may answer more complex business questions, such as '*How'* certain phenomena (or events) occurred within TG contexts. As our understanding of TG deepens, these insights gradually converge into knowledge, which can be used for decision-making. To do so, we have chosen to focus on the analytical approach of Pattern Mining in TG. Pattern Mining techniques for TG allow extracting patterns that combine information from the diverse dimensions of TG (topology, attributes, time) to represent evolution mechanisms present in TG. This extraction is done through the design of mining algorithms.

One of the main challenges in Pattern Mining in TG is to design patterns capturing the maximum information from TG' dimensions to enrich their explanatory power. However, current patterns are limited. They may be not sequential (i.e., describing a sequence of events in order to account for the temporal relationships between events) or complete (i.e., accounting for all the information derived from the diverse evolution types of TG). To the best of our knowledge, no pattern is representative (i.e., it accounts for the evolution of several groups of vertices). In response to these limitations, we have proposed a novel pattern, called *Frequent Sequential Subgraph Evolutions* (FSSE). The advantage of our pattern is to be sequential, complete and representative.

The other main challenge in Pattern Mining in TG is to design mining algorithms minimizing the computational cost. Indeed, the complex data structure of TG requires exploring its different dimensions to extract patterns. In the case of our pattern, the challenge is even more important, as it captures more information from TG dimensions than current patterns. To do so, we have proposed a novel algorithm, the *FSSEMiner.* It enables to extract FSSE patterns from TG. The advantage of our algorithm is to rely on a novel mining strategy we have proposed, the *graph addition.* It allows avoiding repetitive graph traversals to save computation time while guaranteeing the extraction of our complete pattern.

We have carried out experiments on synthetic datasets to evaluate how the computation time of the algorithm FSSEMiner varies according to different graph parameters (timestamps, attributes and number of vertices and edges). We have observed that the algorithm is relatively scalable considering that users will not choose too large graph parameters (more than 14 timestamps and 8 attributes) since this makes the analysis of patterns more difficult. We have also made experiments over two real datasets including disruptive events : (i) a dataset on the US Flights traffic during the Katrina hurricane and (ii) a dataset on the travel flows between cities and COVID cases in China during the

COVID periods. The experiments have shown that the extracted patterns by the FSSEMiner algorithm provide interesting insights to understand the impacts of the events and make more informed decisions.

Chapter 5

Conclusion

Contents

1 Contributions

The digital world provides a representation of real-world entities that are becoming increasingly interconnected (e.g., in social platforms). Data generated by interconnected entities of the real-world has contributed to the emergence of the concept of *Graphs*. Graphs, a collection of vertices connected by edges, naturally represent real-world entities and their relationships. Nevertheless, real-world entities and their relationships evolve continually over time at different levels : (i) entities (or relationships) may appear and disappear over time, (ii) the descriptive characteristics of entities (or relationships) may be added or removed over time and (iii) the value of these descriptive characteristics may change over time.

The temporal evolution of graph data opens new analytical opportunities. However, static graphs are not enough to integrate the concept of temporal evolution in graph data. Thus, this thesis has addressed the problem of enabling analyses on graph data enriched with temporal evolution. This problem induces three main challenges. First, to incorporate static graphs with temporal evolution, it requires determining (i) the levels of the graph subject to the evolution, (ii) the abstraction levels embodied by the graph with temporal evolution in terms of its proximity to a real-world or a technical representation and, (iii) the strategy to manage the changes in graph data. Second, to explore graphs with temporal evolution, it requires enabling users to find information to answer simple business-oriented questions (*'What?', 'Who?', 'Where?', 'When?'*). Third, to go deeper in the exploration of graphs with temporal evolution, it implies to extract knowledge to answer complex business-oriented questions (*'How?'*). Knowledge here is a combination of information pieces from the multiple dimensions of a graph with temporal evolution that serve for understanding some phenomena (or events) and helping decision-making. Facing these three challenges, we have proposed a solution in three parts, as follows.

First, we have proposed a complete management solution for graphs with temporal evolution, from a conceptual model, called *Temporal Graph* (TG), to its implementation. Our conceptual model enables to integrate the concept of temporal evolution in static graphs. It has several advantages compared to existing models. First, it provides a conceptual view to represent closely real-world applications, notably by introducing the concept of states to capture the evolution of entities and relationships. Second, our model associates the concept of temporal evolution at all levels of the graph (topology, attribute set and attribute value) enabling the direct access to the information about changes during analyses. Third, we have proposed an automatic (i.e., no specific development is required) implementation process of our conceptual model into a physical technical environment. It consists of direct mapping rules from our conceptual model to the logical property graph model, which is compatible with several graph-oriented NoSQL data stores. The experimental results have shown that our management solution is (i) feasible, i.e., implementable into graph-oriented NoSQL data stores (here, we chose Neo4j), (ii) usable for business analyses, (iii) efficient in terms of storage and query performance compared to the classic TG models (graph snapshots), and (iv) scalable when the data volume increases. Moreover, our management solution has proven its value in the industry. The concepts of our TG will be used to bring a temporal layer to the graph-oriented software of ACTIVUS Group. Moreover, the Neo4j company has contacted us following the publication of our article Andriamampianina et al. (2022a) to discuss the concepts of our TG and its implementation for their own applications.

Second, we have proposed a querying solution to find information in graphs with temporal evolution to enable users to answer simple *'What?', 'Who?', 'Where?', 'When?'* questions. The advantage of our querying solution compared to current querying solutions for graphs with temporal evolution is to be complete. It starts from a generic framework to an implementation framework. On the one hand, the generic framework focuses on the design of business analyses on a graph with temporal evolution by providing two conceptual operators. The $match_{predicates}$ allows finding time dependent information on the attribute dimension of temporal graph data. The $match_{pattern}$ allows finding time dependent information on the topology dimension of temporal graph data. Unlike current querying solutions, these operators allow to fully manipulate the granularities of the different dimensions of temporal graph data. Moreover, they are based on the business-oriented concepts of our TG model to be more user-oriented. As they are composable, they facilitate the expression of complex queries and their extension to add new functionalities. On the other hand, the implementation framework guarantees that our conceptual operators are directly compatible to different textual query languages for the property graph model (here, query languages of Neo4j and OrientDB). Indeed, we have proposed mapping rules of our conceptual operators to logical operators for querying the property graph model. In a nutshell, our querying solution has the advantage of being independent of a specific implementation but compatible with several implementation alternatives. We have verified through experiments that the querying solution allows effectively applying business analyses on real-world datasets.

Third, we have proposed an analytical approach to extract knowledge in graphs with temporal evolution to enable users to answer *'How ?'* questions. This analytical approach is Pattern Mining. It involves designing (i) a pattern to specify the combination of information pieces from the dimensions of a graph with temporal evolution to be extracted and (ii) an algorithm to extract this pattern. We observed in the literature that existing patterns capture partially the information from the dimensions of graphs with temporal evolution. This hinders the ability to extract complete and meaningful understanding of hidden evolution mechanisms. To do so, we have proposed a new pattern called *Frequent Sequential Subgraph Evolution* (FSSE). It fully captures the information from all dimensions within a graph with temporal evolution. In addition, it is more representative than existing patterns. It allows representing evolution mechanisms that cover several groups of entities instead of a single group as in current patterns. Representative patterns are indeed more useful for users (notably, decision-makers) to obtain a global understanding of evolution trends. Moreover, we have proposed a novel algorithm (*FSSEMiner*) to extract our novel pattern. Since all pattern mining algorithms face the problem of high computational complexity, we have proposed the mining strategy of *graph addition* to reduce the latter. The experiments we conducted have shown that (i) our algorithm is relatively scalable, considering that users will not choose too large datasets since this makes the analysis of patterns more difficult, and (ii) our pattern is useful to understand the impacts of disruptive events in real-world datasets.

This Ph.D. research has been validated through different publications, namely one international journal paper (DKE (Andriamampianina et al., 2022a)), three international conference papers (RCIS (Andriamampianina et al., 2021), CAISE (Andriamampianina et al., 2022b) and PAKDD (Cheng et al., 2023) and one national conference paper (EDA (Andriamampianina et al., 2020)). Table 5.1 summarizes the main topics addressed by each paper. All main topics have been discussed in the previous chapters of the thesis.

Table 5.1: Main topics addressed by the publications during my thesis

		Modelling TG (Chapter 2)	Querying Information in TG (Chapter 3)	Knowledge Discovery in TG (Chapter 4)
International Journals	(Andriamampianina et al., 2022a)	✓		
International Conferences	(Andriamampianina et al., 2021)	✓		
	(Andriamampianina et al., 2022b)		✓	
	(Cheng et al., 2023)			✓
	(Andriamampianina et al., 2023)		(Future work)	
National Conferences	(Andriamampianina et al., 2020)	✓		

2 Future Work

2.1 Short-term plan

2.1.1 Implementation Alternatives of our TG Model and Operators

Currently, we have implemented our solutions into Neo4j and OrientDB data stores. In the short term, we intend to propose several alternatives for the implementation of our conceptual model of TG and our conceptual operators.

As a first step, we will implement our TG model and our conceptual operators in other environments: (i) on-premise graph data stores (e.g., ArangoDB[1]), and (ii) on-cloud graph data stores (e.g., Azure Cosmos DB [2] and AWS Neptune [3]). In the same way, we will experiment alternative mapping rules with Neo4j that have been discussed with R&D engineers from the Neo4j company.

As a second step, we will make new experiments to evaluate the performance of these implementation alternatives (Cheng et al., 2019; Besta et al., 2023b). To do so, we will establish comparison criteria including (i) quantitative criteria such as volumetry, query runtime, (etc.) (ii) qualitative criteria such as usability evaluation through three aspects: effectiveness, efficiency and satisfaction (ISO, 2018). This contributes to the data management layer of the software of ACTIVUS Group to meet the needs of clients on the choice of implementation environments.

2.1.2 Centrality Analysis in Temporal Graph Data

Classically, another analytical approach dedicated to static graphs is using centrality metrics. Centrality metrics enable to identify the most central or influential entities in graphs (Wan et al., 2021; Meng et al., 2022). We propose to extend these concepts to our Temporal Graph. To take into account the evolution of entities' influence, we intend to propose the concept of *Semantic Temporal Centrality* (Andriamampianina et al., 2023). The advantages of this centrality are to include the various types of entities and relationships, as well as temporal evolutions.

[1] https://arangodb.com/

[2] https://learn.microsoft.com/en-us/azure/cosmos-db/introduction

[3] https://aws.amazon.com/neptune/

2.2 Mid-term plan

2.2.1 Updating Temporal Graph Data

We have proposed a modelling of graphs with temporal evolution. A promising perspective is to consider the ingestion of new data, which leads to the evolution of temporal graph data at the topology or attribute levels (Castelltort and Laurent, 2013; Hant et al., 2014; Steer et al., 2020; Besta et al., 2023a). First, we must define an automatic detection strategy of source changes: (i) the addition or deletion of sources, (ii) the modification of sources' schema and (iii) the update of value in sources. Second, we must define an automatic integration of these changes at the levels of topology and attribute. At the topology level, changes consist in the addition or removal of new entities (or relationships) and in the update of their valid times. At the attribute level, changes consist in creating new states of entities (or relationships) triggered by the addition of new attributes of entities (or relationships), the update of their value and valid times. Finally, we must choose a historization process depending on how the data from the sources evolves.

2.2.2 Improvements of Knowledge Discovery in Temporal Graph Data

There are several opportunities to improve the Knowledge Discovery process we proposed. First, the experiments have shown that the algorithm FSSEMiner records exponential execution times when the characteristics (number of vertices/edges/attributes) of a graph with temporal evolution are too large. We plan to make a distributed version of the algorithm to further improve its performance. We will compare the efficiency of large-scales analyses according to different Big Data storage and processing platforms (Gan et al., 2017).

Second, the experiments have shown that the algorithm generates a lot of extracted patterns. In a first step, we will propose more constraints to users to better specify their needs (Robardet, 2009). In a second step, we will propose a post-processing method to reduce the number of FSSE patterns dedicated to final users (Fournier-Viger et al., 2020a). This could facilitate the pattern analysis and interpretation.

Third, currently the FSSE pattern we propose includes the addition and removal of entities' attributes, the update in entities' attribute value and the addition and removal of relationships. To the best of our knowledge, it is the most complete in the literature (Fournier-Viger et al., 2020a). However, we will further improve our pattern by integrating the attributes on relationships and their evolution. The pattern will capture more information, making it more suitable for various applications and enhancing its explanatory power.

2.3 Long-term plan

2.3.1 New Exploration Perspectives

Vizualization of Massive Temporal Graph Data From a visualization point of view, the challenge is to represent the evolution of relationships and entities in a readable and scalable manner. Current techniques need to be extended to offer new solutions for visualizing large amounts of graph data evolving over time. There are several promising lines of research to extend the concepts of: (i) hyperbolic graph and fish-eye views to facilitate descriptive analyses (Miller et al., 2022), (ii) animation-based views in temporal graphs to highlight data evolutions (Beck et al., 2017), (iii) data representation through trajectories describing spatio-temporal movements of entities (Sakr et al., 2022; Godfrid et al., 2022), and (iv) representation of relationships between objects in images (Clément et al., 2016, 2020).

Predictive Analysis In this thesis, we address the descriptive analysis of temporal graph data via a querying solution and an explanatory analysis of temporal graph data via a knowledge discovery solution. To go further in the analysis of temporal graph data, we plan to deal with predictive analysis, i.e., make predictions of future events, trends, or outcomes. There are several promising lines of research on this topic. One promising line of research is graph embedding. It is used to transform a temporal graph into low-dimension vectors, preserving the important structural and semantic information for various predictive tasks (Barros et al., 2021; Wu et al., 2022). Another promising line of research is attention mechanisms in predictive algorithms, which allow assigning different levels of importance of data in temporal graphs when making predictions (Wang et al., 2019).

2.3.2 Temporal Graph in Data Lakes

A long-term perspective would be to integrate this work into a more complex architecture dedicated to data sciences. In particular, numerous works have provided solutions in data lake architectures (Ravat and Zhao, 2019a). Such architectures comprise several data zones (ingestion zone, preparation zone and analysis zone) and a governance zone. For the governance zone, a key element is metadata management (Ravat and Zhao, 2019b), which makes it possible to describe not only the data in different zones, but also the transformation processes (Megdiche et al., 2021) and even the analyses carried out on this data (Dang et al., 2021). Therefore, we will propose new solutions at the levels of metadata and data management. Currently, metadata management solutions are based on graph-based modelling. In this context, one problem is still open: how to manage the temporal evolution of metadata? New research perspectives are therefore (i) to integrate the temporal evolutions in these metamodels and, (ii) to propose new querying and visualization solutions of massive temporal graph data in order to explore the content of a data lake. Another perspective is to integrate a temporal graph as one of the components for data management in a data lake. To combine different storage spaces, we will need to define new data matching or mapping mechanisms that evolve over time (Zhang and Ives, 2020).

Bibliography

Adrian, M. and Jaffri, A. (2022). Market Guide for Graph Database Management Systems.

Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data - SIGMOD '93*, pages 207–216, Washington, D.C., United States. ACM Press.

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the eleventh international conference on data engineering*, pages 3–14. IEEE.

Agrawal, R., Srikant, R., and others (1994). Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499. Santiago, Chile.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843.

Andriamampianina, L., Ravat, F., Song, J., and Vallès-Parlangeau, N. (2020). A generic modelling to capture the temporal evolution in graphs. In *16e journées EDA: Business Intelligence & Big Data (EDA 2020)*, pages 19–32.

Andriamampianina, L., Ravat, F., Song, J., and Vallès-Parlangeau, N. (2021). Towards an Efficient Approach to Manage Graph Data Evolution: Conceptual Modelling and Experimental Assessments. In Cherfi, S., Perini, A., and Nurcan, S., editors, *Research Challenges in Information Science*, pages 471–488. Springer International Publishing.

Andriamampianina, L., Ravat, F., Song, J., and Vallès-Parlangeau, N. (2022a). Graph data temporal evolutions: From conceptual modelling to implementation. *Data & Knowledge Engineering*, 139:102017.

Andriamampianina, L., Ravat, F., Song, J., and Vallès-Parlangeau, N. (2022b). Querying Temporal Property Graphs. In Franch, X., Poels, G., Gailly, F., and Snoeck, M., editors, *Advanced Information Systems Engineering*, volume 13295, pages 355–370. Springer International Publishing, Cham. Series Title: Lecture Notes in Computer Science.

Andriamampianina, L., Ravat, F., Song, J., and Vallès-Parlangeau, N. (2023). Semantic Centrality for Temporal Graphs. In Abelló, A., Vassiliadis, P., Romero, O., Wrembel, R., Bugiotti, F., Gamper, J., Vargas Solar, G., and Zumpano, E., editors, *New Trends in Database and Information Systems*, volume 1850, pages 163–173. Springer Nature Switzerland, Cham. Series Title: Communications in Computer and Information Science.

Angles, R. (2018). The Property Graph Database Model. In *AMW*.

Angles, R., Arenas, M., Barcelo, P., Boncz, P., Fletcher, G., Gutierrez, C., Lindaaker, T., Paradies, M., Plantikow, S., Sequeda, J., van Rest, O., and Voigt, H. (2018). G-CORE: A Core for Future Graph Query Languages. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1421–1432, Houston TX USA. ACM.

Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J., and Vrgoč, D. (2017). Foundations of Modern Query Languages for Graph Databases. *ACM Computing Surveys*, 50(5):68:1–68:40.

Angles, R. and Gutierrez, C. (2008). Survey of graph database models. *ACM Computing Surveys*, 40(1):1–39.

Angles, R. and Gutierrez, C. (2018). An introduction to Graph Data Management. *arXiv:1801.00036 [cs]*, pages 1–32. arXiv: 1801.00036.

Aslay, C., Nasir, M. A. U., De Francisci Morales, G., and Gionis, A. (2018). Mining Frequent Patterns in Evolving Graphs. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 923–932. ACM.

Barros, C. D., Mendonça, M. R., Vieira, A. B., and Ziviani, A. (2021). A survey on embedding dynamic graphs. *ACM Computing Surveys (CSUR)*, 55(1):1–37. Publisher: ACM New York, NY.

Beck, F., Burch, M., Diehl, S., and Weiskopf, D. (2017). A Taxonomy and Survey of Dynamic Graph Visualization. *Computer Graphics Forum*, 36(1):133–159.

Bellinger, G., Castro, D., and Mills, A. (2004). Data, information, knowledge, and wisdom.

Besta, M., Fischer, M., Kalavri, V., Kapralov, M., and Hoefler, T. (2023a). Practice of Streaming Processing of Dynamic Graphs: Concepts, Models, and Systems. *IEEE Transactions on Parallel and Distributed Systems*, 34(6):1860–1876.

Besta, M., Gerstenberger, R., Peter, E., Fischer, M., Podstawski, M., Barthels, C., Alonso, G., and Hoefler, T. (2023b). Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. *ACM Computing Surveys*, 56(2):1–40. Publisher: ACM New York, NY.

Bohlen, M., Busatto, R., and Jensen, C. (1998). Point-versus interval-based temporal data models. In *Proceedings 14th International Conference on Data Engineering*, pages 192–200, Orlando, FL, USA. IEEE Comput. Soc.

Campos, A., Mozzino, J., and Vaisman, A. (2016). Towards Temporal Graph Databases. *arXiv:1604.08568 [cs]*. arXiv: 1604.08568.

Castelltort, A. and Laurent, A. (2013). Representing history in graph-oriented nosql databases: A versioning system. In *Eighth International Conference on Digital Information Management (ICDIM 2013)*, pages 228–234. IEEE.

Cattuto, C., Quaggiotto, M., Panisson, A., and Averbuch, A. (2013). Time-varying social networks in a graph database: a Neo4j use case. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, pages 1–6. Association for Computing Machinery.

Cheng, Y., Ding, P., Wang, T., Lu, W., and Du, X. (2019). Which Category Is Better: Benchmarking Relational and Graph Database Management Systems. *Data Science and Engineering*, 4(4):309–322.

Cheng, Z., Andriamampianina, L., Ravat, F., Song, J., Vallès-Parlangeau, N., Fournier-

Viger, P., and Selmaoui-Folcher, N. (2023). Mining Frequent Sequential Subgraph Evolutions in Dynamic Attributed Graphs. In Kashima, H., Ide, T., and Peng, W.-C., editors, *Advances in Knowledge Discovery and Data Mining*, volume 13936, pages 66–78. Springer Nature Switzerland, Cham. Series Title: Lecture Notes in Computer Science.

Cheng, Z., Flouvat, F., and Selmaoui-Folcher, N. (2017). Mining Recurrent Patterns in a Dynamic Attributed Graph. In *Advances in Knowledge Discovery and Data Mining*, volume 10235, pages 631–643. Springer.

Clément, M., Kurtz, C., and Wendling, L. (2020). Fuzzy directional enlacement landscapes for the evaluation of complex spatial relations. *Pattern Recognition*, 101:107185.

Clément, M., Poulenard, A., Kurtz, C., and Wendling, L. (2016). Directional enlacement histograms for the description of complex spatial configurations between objects. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2366–2380.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2022). *Introduction to algorithms*. MIT press.

Dang, V.-N., Zhao, Y., Megdiche, I., and Ravat, F. (2021). A Zone-Based Data Lake Architecture for IoT, Small and Big Data. In *25th International Database Engineering & Applications Symposium (IDEAS 2021)*.

Debrouvier, A., Parodi, E., Perazzo, M., Soliani, V., and Vaisman, A. (2021). A model and query language for temporal graph databases. *The VLDB Journal*, 30(5):825–858.

Desmier, E., Plantevit, M., Robardet, C., and Boulicaut, J.-F. (2012). Cohesive co-evolution patterns in dynamic attributed graphs. In *International Conference on Discovery Science*, pages 110–124. Springer.

Desmier, E., Plantevit, M., Robardet, C., and Boulicaut, J.-F. (2013). Trend mining in dynamic attributed graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 654–669. Springer.

Dignös, A., Böhlen, M. H., and Gamper, J. (2012). Temporal alignment. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 433–444, New York, NY, USA. Association for Computing Machinery.

Fiedler, M. and Borgelt, C. (2007). Support computation for mining frequent subgraphs in a single graph. In *MLG*.

Fournier-Viger, P., Cheng, C., Cheng, Z., Lin, J. C.-W., and Selmaoui-Folcher, N. (2019). Mining significant trend sequences in dynamic attributed graphs. *Knowledge-Based Systems*, 182:104797.

Fournier-Viger, P., He, G., Cheng, C., Li, J., Zhou, M., Lin, J. C.-W., and Yun, U. (2020a). A survey of pattern mining in dynamic graphs. *WIREs Data Mining and Knowledge Discovery*, 10(6):e1372.

Fournier-Viger, P., He, G., Lin, J. C.-W., and Gomes, H. M. (2020b). Mining attribute evolution rules in dynamic attributed graphs. In *International Conference on Big Data*

Analytics and Knowledge Discovery, pages 167–182. Springer.

Fournier-Viger, P., Lin, J. C.-W., Kiran, R. U., Koh, Y. S., and Thomas, R. (2017). A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77.

Gan, W., Lin, J. C.-W., Chao, H.-C., and Zhan, J. (2017). Data mining in distributed environment: a survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(6):e1216. Publisher: Wiley Online Library.

Gan, W., Lin, J. C.-W., Fournier-Viger, P., Chao, H.-C., and Yu, P. S. (2019). A Survey of Parallel Sequential Pattern Mining. *ACM Transactions on Knowledge Discovery from Data*, 13(3):1–34. arXiv:1805.10515 [cs].

Gandhi, S. and Simmhan, Y. (2020). An Interval-centric Model for Distributed Computing over Temporal Graphs. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1129–1140. IEEE.

Ghrab, A., Romero, O., Jouili, S., and Skhiri, S. (2018). Graph BI & Analytics: Current State and Future Challenges. In Ordonez, C. and Bellatreche, L., editors, *Big Data Analytics and Knowledge Discovery*, volume 11031, pages 3–18. Springer International Publishing, Cham. Series Title: Lecture Notes in Computer Science.

Godfrid, J., Radnic, P., Vaisman, A., and Zimányi, E. (2022). Analyzing public transport in the city of Buenos Aires with MobilityDB. *Public Transport*, 14(2):287–321.

Hant, W., Miao, Y., Li, K., Wu, M., Yang, F., Zhou, L., Prabhakaran, V., Chen, W., and Chen, E. (2014). Chronos: a graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems - EuroSys '14*, pages 1–14, Amsterdam, The Netherlands. ACM Press.

Holme, P. (2015). Modern temporal network theory: A colloquium. *The European Physical Journal B*, 88(9):234.

Holme, P. and Saramäki, J. (2012). Temporal networks. *Physics reports*, 519(3):97–125. Publisher: Elsevier.

Huang, H., Song, J., Lin, X., Ma, S., and Huai, J. (2016). TGraph: A Temporal Graph Data Management System. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 2469–2472. ACM.

ISO (2018). ISO 9241-11:2018(en), Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts.

Jensen, C. S., Clifford, J., Gadia, S. K., Segev, A., and Snodgrass, R. T. (1992). A glossary of temporal database concepts. *ACM SIGMOD Record*, 21(3):35–43.

Jian Pei, Jiawei Han, Mortazavi-Asl, B., Jianyong Wang, Pinto, H., Qiming Chen, Dayal, U., and Mei-Chun Hsu (2004). Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440.

Johnston, T. and Weis, R. (2010). A Brief History of Temporal Data Management. In

Managing Time in Relational Databases, pages 11–25. Elsevier.

Kaytoue, M., Pitarch, Y., Plantevit, M., and Robardet, C. (2014). Triggering patterns of topology changes in dynamic graphs. In *International Conference on Advances in Social Networks Analysis and Mining*, pages 158–165. IEEE.

Khurana, U. and Deshpande, A. (2013). Efficient snapshot retrieval over historical graph data. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 997–1008. IEEE.

Khurana, U. and Deshpande, A. (2016). Storing and Analyzing Historical Graph Data at Scale. In *EDBT*.

Koloniari, G., Souravlias, D., and Pitoura, E. (2013). On Graph Deltas for Historical Queries. *CoRR*, abs/1302.5549.

Kosmatopoulos, A., Giannakopoulou, K., Papadopoulos, A. N., and Tsichlas, K. (2016). An Overview of Methods for Handling Evolving Graph Sequences. In Karydis, I., Sioutas, S., Triantafillou, P., and Tsoumakos, D., editors, *Algorithmic Aspects of Cloud Computing*, volume 9511, pages 181–192. Springer International Publishing.

Lassila, O. and Swick, R. R. (1998). Resource description framework (RDF) model and syntax specification.

Latapy, M., Viard, T., and Magnien, C. (2018). Stream Graphs and Link Streams for the Modeling of Interactions over Time. *Social Networks Analysis and Mining*, 8(1):61:1–61:29.

Luna, J. M., Fournier-Viger, P., and Ventura, S. (2019). Frequent itemset mining: A 25 years review. *WIREs Data Mining and Knowledge Discovery*, 9(6).

Madan, A., Cebrian, M., Moturu, S., Farrahi, K., and Pentland, A. S. (2012). Sensing the "Health State" of a Community. *IEEE Pervasive Computing*, 11(4):36–45.

Massri, M., Miklos, Z., Raipin, P., and Meye, P. (2022). Clock-G: A temporal graph management system with space-efficient storage technique. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pages 2263–2276, Kuala Lumpur, Malaysia. IEEE.

Megdiche, I., Ravat, F., and Zhao, Y. (2021). Metadata Management on Data Processing in Data Lakes. In Bureš, T., Dondi, R., Gamper, J., Guerrini, G., Jurdziński, T., Pahl, C., Sikora, F., and Wong, P. W., editors, *SOFSEM 2021: Theory and Practice of Computer Science*, volume 12607, pages 553–562. Springer International Publishing, Cham. Series Title: Lecture Notes in Computer Science.

Meng, Y., Qi, Q., Liu, J., and Zhou, W. (2022). Dynamic Evolution Analysis of Complex Topology and Node Importance in Shenzhen Metro Network from 2004 to 2021. *Sustainability*, 14(12):7234. Number: 12 Publisher: Multidisciplinary Digital Publishing Institute.

Miller, J., Kobourov, S., and Huroyan, V. (2022). Browser-based Hyperbolic Visualization of Graphs. arXiv:2205.08028 [cs].

Moffitt, V. Z. and Stoyanovich, J. (2017a). Temporal graph algebra. In *Proceedings of The 16th International Symposium on Database Programming Languages*, DBPL '17, pages 1–12, Munich, Germany. Association for Computing Machinery.

Moffitt, V. Z. and Stoyanovich, J. (2017b). Towards sequenced semantics for evolving graphs. In *EDBT*, pages 446–449.

Nicosia, V., Tang, J., Mascolo, C., Musolesi, M., Russo, G., and Latora, V. (2013). Graph Metrics for Temporal Networks. *arXiv:1306.0493 [physics]*, pages 15–40. arXiv: 1306.0493.

Pei, J., Han, J., and Wang, W. (2007). Constraint-based sequential pattern mining: the pattern-growth methods. *Journal of Intelligent Information Systems*, 28(2):133–160.

Pernelle, N., Saïs, F., Mercier, D., and Thuraisamy, S. (2016). RDF data evolution: efficient detection and semantic representation of changes. *Semantic Systems-SEMANTiCS2016*, page 4.

Ramesh, S., Baranawal, A., and Simmhan, Y. (2020). A Distributed Path Query Engine for Temporal Property Graphs. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 499–508. IEEE.

Ravat, F., Song, J., Teste, O., and Trojahn, C. (2019). Improving the performance of querying multidimensional RDF data using aggregates. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, pages 2275–2284. Association for Computing Machinery.

Ravat, F. and Zhao, Y. (2019a). Data Lakes: Trends and Perspectives. In Hartmann, S., Küng, J., Chakravarthy, S., Anderst-Kotsis, G., Tjoa, A. M., and Khalil, I., editors, *Database and Expert Systems Applications*, Lecture Notes in Computer Science, pages 304–313, Cham. Springer International Publishing.

Ravat, F. and Zhao, Y. (2019b). Metadata Management for Data Lakes. In Welzer, T., Eder, J., Podgorelec, V., Wrembel, R., Ivanović, M., Gamper, J., Morzy, M., Tzouramanis, T., Darmont, J., and Kamišalić Latifić, A., editors, *New Trends in Databases and Information Systems*, volume 1064, pages 37–44. Springer International Publishing, Cham. Series Title: Communications in Computer and Information Science.

Reinsel, D., Gantz, J., and Rydning, J. (2017). Data age 2025: The evolution of data to life-critical.

Ren, C., Lo, E., Kao, B., Zhu, X., and Cheng, R. (2011). On querying historical evolving graph sequences. *Proceedings of the VLDB Endowment*, 4(11):726–737.

Robardet, C. (2009). Constraint-based pattern mining in dynamic graphs. In *2009 ninth IEEE international conference on data mining*, pages 950–955. IEEE.

Rossi, R. A., Gallagher, B., Neville, J., and Henderson, K. (2013). Modeling dynamic behavior in large evolving graphs. In *Proceedings of the sixth ACM international conference on Web search and data mining - WSDM '13*, pages 667–676. ACM Press.

Rost, C., Gomez, K., Täschner, M., Fritzsche, P., Schons, L., Christ, L., Adameit, T., Junghanns, M., and Rahm, E. (2021). Distributed temporal graph analytics with GRADOOP. *The VLDB Journal*.

Roussakis, Y., Chrysakis, I., Stefanidis, K., Flouris, G., and Stavrakas, Y. (2015). A flexible framework for understanding the dynamics of evolving RDF datasets. In *International Semantic Web Conference*, pages 495–512. Springer.

Rowley, J. (2007). The wisdom hierarchy: representations of the DIKW hierarchy. *Journal of Information Science*, 33(2):163–180.

Sakr, M., Zimanyi, E., Vaisman, A., and Bakli, M. (2022). User-centered road network traffic analysis with MobilityDB. *Transactions in GIS*, 27.

Sharma, C., Sinha, R., and Johnson, K. (2021). Practical and comprehensive formalisms for modeling contemporary graph query languages. *Info. Systems*, page 101816.

Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In *Advances in Database Technology—EDBT'96: 5th International Conference on Extending Database Technology Avignon, France, March 25–29, 1996 Proceedings 5*, pages 1–17. Springer.

Steer, B., Cuadrado, F., and Clegg, R. (2020). Raphtory: Streaming analysis of distributed temporal graphs. *Future Generation Computer Systems*, 102:453–464.

Telikani, A., Gandomi, A. H., and Shahbahrami, A. (2020). A survey of evolutionary computation for association rule mining. *Information Sciences*, 524:318–352.

Tian, Y. (2022). The World of Graph Databases from An Industry Perspective. arXiv:2211.13170 [cs].

Van Rest, O., Hong, S., Kim, J., Meng, X., and Chafi, H. (2016). PGQL: a property graph query language. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems - GRADES '16*, pages 1–6, Redwood Shores, California. ACM Press.

Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., and Wilkins, D. (2010). A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th Annual Southeast Regional Conference on - ACM SE '10*, page 1. ACM Press.

Wan, Z., Mahajan, Y., Kang, B. W., Moore, T. J., and Cho, J.-H. (2021). A Survey on Centrality Metrics and Their Network Resilience Analysis. *IEEE Access*, 9:104773–104819.

Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., and Yu, P. S. (2019). Heterogeneous Graph Attention Network. In *The World Wide Web Conference*, WWW '19, pages 2022–2032, New York, NY, USA. Association for Computing Machinery.

Wang, X. S., Jajodia, S., and Subrahmanian, V. (1993). Temporal modules: An approach toward federated temporal databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 227–236.

Wu, T., Khan, A., Yong, M., Qi, G., and Wang, M. (2022). Efficiently embedding dynamic knowledge graphs. *Knowledge-Based Systems*, 250:109124.

Xiangyu, L., Yingxiao, L., Xiaolin, G., and Zhenhua, Y. (2020). An Efficient Snapshot Strategy for Dynamic Graph Storage Systems to Support Historical Queries. *IEEE Access*, 8:90838–90846.

Yang, Y., Yu, J. X., Gao, H., Pei, J., and Li, J. (2014). Mining most frequently changing component in evolving graphs. *World Wide Web*, 17(3):351–376.

Zaki, A., Attia, M., Hegazy, D., and Amin, S. (2016). Comprehensive Survey on Dynamic Graph Models. *International Journal of Advanced Computer Science and Applications*, 7(2).

Zaki, M. J. (2001). Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42:31–60.

Zhang, F., Wang, K., Li, Z., and Cheng, J. (2019). Temporal Data Representation and Querying Based on RDF. *IEEE Access*, 7:85000–85023. Conference Name: IEEE Access.

Zhang, Y. and Ives, Z. G. (2020). Finding Related Tables in Data Lakes for Interactive Data Science. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1951–1966, Portland OR USA. ACM.

Zhao, A., Liu, G., Zheng, B., Zhao, Y., and Zheng, K. (2020). Temporal paths discovery with multiple constraints in attributed dynamic graphs. *World Wide Web*, 23(1):313–336.