UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Graph data warehousing

Amine Ghrab

Graph Data Warehousing

Ph.D. Dissertation

Amine Ghrab

Dissertation submitted in September, 2020

A thesis submitted to the Faculty of Engineering at Université Libre De Bruxelles (ULB) and the Barcelona School of Informatics at Universitat Politècnica de Catalunya, BarcelonaTech (UPC), in partial fulfillment of the requirements within the scope of the IT4BI-DC programme for the joint Ph.D. degree in computer science. The thesis is not submitted to any other organization at the same time.

Thesis submitted:	August, 2020
UPC Ph.D. Supervisor:	Prof. Oscar Romero Universitat Politècnica de Catalunya, BarcelonaTech, Spain
ULB Ph.D. Supervisor:	Prof. Esteban Zimányi Université Libre de Bruxelles, Brussels, Belgium
EURA NOVA Supervisor:	Sabri Skhiri EURA NOVA, Mont-Saint-Guibert, Belgium

ISSN: xxxx-xxxx
ISBN: xxx-xx-xxxx-xxx-x

Printed in Belgium, 2020

Curriculum Vitae

Amine Ghrab

Amine Ghrab graduated in *Computer Science Engineering* in October 2011, at the *Faculté des Sciences de Tunis*, *Université Tunis El Manar*.

In 2013, Amine joined the Erasmus Mundus Joint Doctorate program of *Information Technologies for Business Intelligence, Doctoral College* (IT4BI-DC). His PhD studies were under the supervision of Prof. Esteban Zimányi at the Department of Computer and Decision Engineering, at Université Libre de Bruxelles, in cohort with Universitat Politècnica de Catalunya (UPC) under the supervision of Prof. Oscar Romero at the department of Service and Information System Engineering (ESSI). His research interests mainly fall at the intersection of business intelligence and graph fields, namely: *Graph Data management*, *Graph Mining*, *Data warehousing*, *Multidimensional Modeling*, *Distributed Graph Processing*.

As part of his joint PhD studies, Amine performed two research stays at *Universitat Politècnica de Catalunya*, his host university, working with professor Oscar Romero (October - December 2012, and September - December 2013). He actively participated in five summer schools and contributed to the organization of three workshops on stream processing.

While doing his PhD he has published 7 peer-reviewed publications, including 1 journal paper, 3 research track full conference papers and 3 workshop papers. Besides that, he has also co-advised 9 master thesis and was invited by multiple research groups to give seminars on the topic of graph warehousing.

In parallel to his PhD, he was employed as R&D Engineer and consultant at EURA NOVA SA. He participated as an R&D architect in the industrial research projects BI on Graph, Jericho and ASGARD.

Abstract

Over the last decade, we have witnessed the emergence of networks in a wide spectrum of application domains, ranging from social and information networks to biological and transportation networks. Graphs provide a solid theoretical foundation for modeling complex networks and revealing valuable insights from both the network structure and the data embedded within its entities. As the business and social environments are getting increasingly complex and interconnected, graphs became a widespread abstraction at the core of the information infrastructure supporting those environments. Modern information systems consist of a large number of sophisticated and interacting business entities that naturally form graphs. In particular, integrating graphs into data warehouse systems received a lot of interest from both academia and industry. Indeed, data warehouses are the central enterprise's information repository and are critical for proper decision support and future planning. Graph warehousing is emerging as the field that extends current information systems with graph management and analytics capabilities. Many approaches were proposed to address the graph data warehousing challenge. These efforts laid the foundation for multidimensional modeling and analysis of graphs. However, most of the proposed approaches partially tackle the graph warehousing problem by being restricted to simple abstractions such as homogeneous graphs or ignoring important topics such as multidimensional integrity constraints and dimension hierarchies.

In this dissertation, we conduct a systematic study of the graph data warehousing topic and address the key challenges of database and multidimensional modeling of graphs. We first propose GRAD, a new graph database model tailored for graph warehousing and OLAP analytics. GRAD aims to provide analysts with a set of simple, well-defined, and adaptable conceptual components to support rich semantics and perform complex analysis on graphs. Then, we define the multidimensional concepts for heterogeneous attributed graphs and highlight the new types of measures that could be derived. We project this multidimensional model on property graphs and explore how to extract the candidate multidimensional concepts and build graph cubes. Then, we extend the multidimensional model by integrating

GRAD and show how GRAD facilitates multidimensional graph modeling, and enables supporting dimension hierarchies and building new types of OLAP cubes on graphs. Afterward, we present TopoGraph, a graph data warehousing framework that extends current graph warehousing models with new types of cubes and queries combining graph-oriented and OLAP querying. TopoGraph goes beyond traditional OLAP cubes, which process value-based grouping of tables, by considering also the topological properties of the graph elements. And it goes beyond current graph warehousing models by proposing new types of graph cubes. These cubes embed a rich repertoire of measures that could be represented with numerical values, with entire graphs, or as a combination of them. Finally, we propose an architecture of the graph data warehouse and describe its main building blocks and the remaining gaps. The various components of the graph warehousing framework can be effectively leveraged as a foundation for designing and building industry-grade graph data warehouses.

We believe that our research in this thesis brings us a step closer towards a better understanding of graph warehousing. Yet, the models and framework we proposed are the tip of the iceberg. The marriage of graph and warehousing technologies will bring many exciting research opportunities, which we briefly discuss at the end of the thesis.

Acknowledgments

First of all, I want to express my gratitude to my research director Sabri Skhiri for giving me the opportunity to work on this thesis, for his continuous support, for his availability and inspiration. He was a great source of encouragement. His talent, self-discipline and hardworking inspired me to keep pursuing this PhD. Special thanks goes to my directors Eric and Hervé, this thesis would never have happened without their support.

I sincerely thank my advisor Oscar Romero. His guidance over the course of this thesis greatly helped me. His patience helped me work my way through tough times. For making the whole experience instructive and thoroughly enjoyable, thank you!

I would like to thank my advisor Esteban Zimányi for the numerous interesting discussions and fruitful collaborations.

Thanks to all my colleagues in the Department of Computer & Decision Engineering (CoDE) at the Université Libre de Bruxelles for their help and sympathy. Thanks to all who kindly welcomed me during my time in Barcelona, particularly from the Department Dept. d'Enginyeria de Serveis i Sistemes d'Informació (ESSI) at the Universitat Politècnica de Catalunya.

During the long time I spent in Belgium and Spain, I met many people who touched my life and made these places feel like home. In particular, I feel fortunate to have met Salim, Maher, Mennan and Besim. I would like to thank all my colleagues at the university and EURA NOVA for the stimulating environment you created. Our collaboration over the years was enjoyable and fruitful in several ways.

On a personal note, I want above all to thank my parents Mohamed and Emna, for being there for me, unconditionally and endlessly. I would like to thank my brothers, my sisters and all my family and friends, for their love, support and encouragement over the years. The last word will be for my daughter, my son and my wife. Thank you for being and for always being there for me.

This thesis has been partially funded by EURANOVA SA and the Wallonia Region in Belgium (Grant FIRST-ENTERPRISES N° 6850 and Convention n°7441).

Contents

Thesis Details

Thesis Title:	Graph Data Warehousing
Ph.D. Student:	Amine Ghrab
Supervisors:	Prof. Esteban Zimányi, Université Libre de Bruxelles, Brussels, Belgium (ULB Supervisor)
	Prof. Oscar Romero, Universitat Politècnica de Catalunya, BarcelonaTech (UPC Supervisor)

This PhD thesis is an industrial collaboration with EURANOVA, Mont-Saint-Guibert, Belgium. Mr. Sabri Skhiri is the responsible and supervisor of this thesis at EURANOVA.
The main body of this thesis consists of the following papers.

[1] Amine Ghrab, Sabri Skhiri, Salim Jouili, Esteban Zimányi: *An Analytics-Aware Conceptual Model for Evolving Graphs*. DaWaK 2013: 1-12 [52].

[2] Amine Ghrab, Oscar Romero, Sabri Skhiri, Alejandro A. Vaisman, Esteban Zimányi: *A Framework for Building OLAP Cubes on Graphs*. ADBIS 2015: 92-105 [49].

[3] Amine Ghrab, Oscar Romero, Sabri Skhiri, Alejandro Vaisman, Esteban Zimányi. *GRAD: On Graph Database Modeling*. CoRR abs/1602.00503 (2016) [50].

[4] Amine Ghrab, Oscar Romero, Salim Jouili, Sabri Skhiri: *Graph BI & Analytics: Current State and Open Challenges*. DaWaK 2018: 1-12, 2018 [48].

[5] Amine Ghrab, Oscar Romero, Sabri Skhiri, Esteban Zimányi: *TopoGraph: An End-To-End Framework to Build and Analyze Graph Cubes*. Information Systems Frontiers (2020) [51].

In addition to the main body papers, the following peer-reviewed workshop publications have also been published in collaboration with master students supervised by the PhD candidate:

[6] Benoit Denis, Amine Ghrab, Sabri Skhiri: *A Distributed Approach for Graph-oriented Multidimensional Analysis*. BigData 2013: 9-16 [42].

[7] Florian Demesmaeker, Amine Ghrab, Siegfried Nijssen, Sabri Skhiri: *Discovering Interesting Patterns in Large Graph Cubes*. BigData 2017: 3322-3331 [41].

[8] Muaz Twaty, Amine Ghrab, Sabri Skhiri. *GraphOpt: a Framework for Automatic Parameters Tuning of Graph Processing Frameworks*. BigData 2019 [126].

This thesis has been submitted for assessment in partial fulfillment of the PhD degree. The thesis is based on the submitted or published scientific papers which are listed above. Parts of the papers are used directly or indirectly in the extended summary of the thesis. The thesis is not in its present form acceptable for open publication but only in limited and closed circulation as copyright may not be ensured.

Chapter 1

Introduction

1 Motivation

Graphs are fundamental and widespread structures that provide an intuitive abstraction for the modeling and analysis of complex, heterogeneous, and highly interconnected data. Graph models are one of the few models that consider relationships as a first-class citizen. Thus, putting an equal focus on the entities of a given domain and the connections between those entities. They have the benefit of revealing valuable insights from content-based and topological properties of data. The great expressive power of graphs, along with their solid mathematical background, encourages their use for modeling domains having complex structural relationships. Large complex graphs have emerged in various business and scientific domains. Newman identifies four categories of graphs, namely social networks, information networks, technological networks, and biological networks [95]. For example, in social networks, graphs could be used for modeling relationships between users and mining uncovered structural information such as communities and influencers [77]. Citation networks [104] and the world wide web [23] are the major examples of information networks. In technological networks, graphs could be used by a telecommunication operator to detect the central nodes of the network infrastructure which helps to optimize the routing and load balancing across the infrastructure. In biology, graphs are used for modeling metabolic pathways, genetic regulations, and protein interactions [97].

Due to the graph model characteristics and its wide range of applications, graph analysis is being considered as *"possibly the single most effective competitive differentiator for organizations pursuing data-driven operations and decisions after the design of data capture"* by leading industrial groups such as Gartner, Inc., a research and advisory firm [45]. It is clear that the topological properties of graphs are of a big potential to decision-making systems, and in particular to Business Intelligence (BI) systems. They supply these systems with a new class of complex structural business facts and measures that could be explored for making more accurate decisions in data-driven organizations. Traditional BI systems, and particularly data warehouses, were designed to support relational data management and analysis, which assumes a pre-defined fixed schema and relational structures. However, due to the fundamental difference between graph and relational model, the existing relational systems are not suitable for efficient graph analysis. The structure-driven management and analytics of graph data call for the development of novel data models, query processing paradigms, and storage techniques. Graph BI is therefore emerging as the BI field that extends current BI systems with graph management and analysis capabilities. It enables graph-based insights such as detection of popular users or communities in social networks, or revealing hidden interaction patterns in financial networks. Graph BI can help address the above-mentioned big data applications since (1) data are interconnected in complex ways, but graphs can help reduce this complexity with intuitive data models and queries, (2) the data size is large, but data warehouses and OLAP analysis are suitable for storage, organization, synthesis and analysis of large volumes of data, and (3) graph mining extends traditional techniques by including the discovery of the topological properties, thus characterizing more precisely business applications.

Therefore, as motivated by multiple research lines [29, 43, 83], this thesis aims to extend current BI and analytics systems to efficiently support warehousing [38], processing [120], mining [119] and OLAP analysis [28] of the graph structural and content-based information. Figure 1.1 provides an overview of the different components of the envisioned graph BI system. While adopting a similar template as the traditional BI systems (i.e., it preserves the familiar data analytics workflow), graph BI extends current systems with graph-aware components that deliver graph-derived insights.

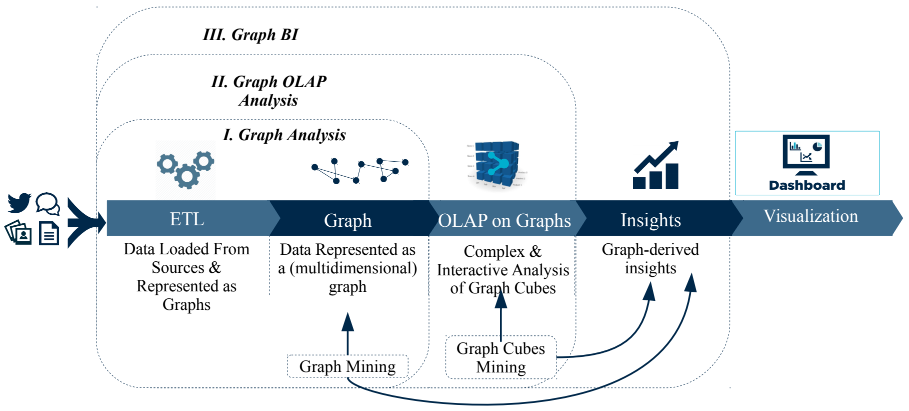


Fig. 1.1: Graph Business Intelligence

2 Background

2.1 BI and Data Warehousing

Business Intelligence (BI) provides tools and techniques for the management and analysis of an organization's data. BI processes enable transforming raw data into valuable information used by managers in strategic decision-making. An organization's data is often stored in a data warehouse, which is the reference information repository. In the data warehouse, operational data is integrated and organized to fit analysis purposes. A common analysis technique in data warehouses is On-Line Analytical Processing (OLAP). OLAP analysis enables complex and interactive querying of large data sets. The data analysis follows a fact/dimension dichotomy to study a business fact according to a set of factors. The analyzed fact is composed of a set of measures placed in the so-called OLAP cubes, and then examined from multiple perspectives and at multiple aggregation levels using the dimensions which are the axes of the cube.
This section reviews the fundamental BI and warehousing concepts used in the thesis.

Business Intelligence

Many definitions were proposed in the literature for Business Intelligence (BI). The term Business Intelligence was defined in 1958 by Hans Peter Luhn [88]. He defined "Business" as *"a collection of activities carried on for whatever purpose, be it science, technology, commerce, industry, law, government, defense, et cetera"* and "Intelligence" as *"the ability to apprehend the interrelationships of* *presented facts in such a way as to guide actions towards the desired goal".* Rizzi defines BI as "*a set of tools and techniques that enable a company to transform its business data into timely and accurate information for the decisional process, to be made available to the right persons in the most suitable form*" [107]. Therefore, the main goal of BI is to supply decision-makers with the right amount of information to help them get a better understanding of the key factors that influence their business's evolution. Thus enabling them to make more informed decisions and increase their organization's competitiveness. To this end, BI systems offer a set of processes for transforming raw data into useful information and actionable knowledge. They provide the required tools for efficient storage, interactive querying, and easy retrieval of information from huge volumes of data.

Traditional BI systems assumed the data to be complete, consistent, and carefully preprocessed and integrated. However, with the increasing market pressure and recent technological advances, BI systems have to evolve from the traditional assumptions to the Big Data environments. Within this emerging setting, the volume, velocity, and variety of data increase tremendously. In addition to data change, new business requirements are emerging such as interactive and discovery-driven analyses, predictive analytics, and integration of social media and web data. This evolution of the BI market provided new opportunities but poses new challenges that call for novel theoretical foundations and enabling technological capabilities. Multiple BI challenges are being tackled from both academic and industrial perspectives. This line of work, sometimes denoted as BI 2.0, tackles some emerging research fields such as real-time BI, collaborative BI, and pervasive BI, as outlined by Aufaure et al. [11]. In this thesis we focus on graph BI to extend current BI tools with graph analysis capabilities, therefore enabling BI analysis of graph-based insights such as the popularity of a user in a given network, or the shortest distance between a pair of locations. This line of work requires designing new data models, query languages, and building the corresponding processing frameworks [11, 38, 98].

Data Warehouse

The data warehouse is considered as the core storage repository on top of which BI operations are mainly executed. Bill Inmon defined the data warehouse as *" a subject-oriented, integrated, time-variant and non-volatile collection of data used in strategic decision-making"* [65]. Thus, a data warehouse provides a central, historical, integrated, and consistent repository of the organization's data.

Data is gathered and integrated from multiple data sources such as spreadsheets, customer relationship management (CRM), and enterprise resource planning (ERP), but mostly from operational databases. The operational data

is periodically fed to refresh the data warehouse. Two main approaches are often used to build a data warehouse: (1) Extract-Transform-Load (ETL) and (2) Extract-Load-Transform (ELT) [127]. In both approaches, data is first extracted from different data sources. In ETL, data is first loaded into a staging database where a set of transformations is first applied to cleanse the data before loading it into the data warehouse. This guarantees the trustworthiness, access efficiency, reliability, and integration of the data. However, this approach assumes that all transformations are being considered in advance in the ETL process. Adding new data or transformations requires the redesign and development of the process. ETL is a pull-based process ran periodically to refresh the data warehouse with required information from underlying sources. The ETL is a critical and heavy step, a set of approaches was developed to speed it up and get closer to real-time data warehousing [132]. The second approach, ELT, is an alternative approach that consists in pushing the transformation phase to the data mart server engine rather than on the ETL middle-ware. The data is then moved only once over the network. This enables copying all available operational data into the data warehouse, which includes data that might be useful in future analysis scenarios. Another approach is the Capture-Transform-Flow (CTL), which flows data in real-time into a continuously fed refreshed data warehouse, following a publisher-subscriber model. Once the data is available in the data warehouse, it is loaded within correspondent data marts. Data marts are the underlying departmental stores of the data warehouse.
The historical and integrated view provided by the data warehouses makes it the backbone for supporting reporting, that presents quantitative data in a report-oriented format using charts, numbers and graphics, and data mining algorithms, executed to identify hidden patterns based on statistical methods such as association analysis, classification, clustering. Figure 1.2 illustrates the phases of the traditional data warehousing process from data collection, through ETL, then storage and management, to cube construction and analysis.

Multidimensional Modeling

From a data modeling perspective, data is organized according to the multidimensional model [2, 27, 66]. The main concepts of the multidimensional model are described in this section.

Fact and Measures Facts are the objects that represent the subjects of the desired analyses, i.e., the interesting entities, events, or processes, that are to be analyzed to better understand their behavior. In the multidimensional model, a measure is the basic unit of data that is placed for analysis in the multidimensional space.

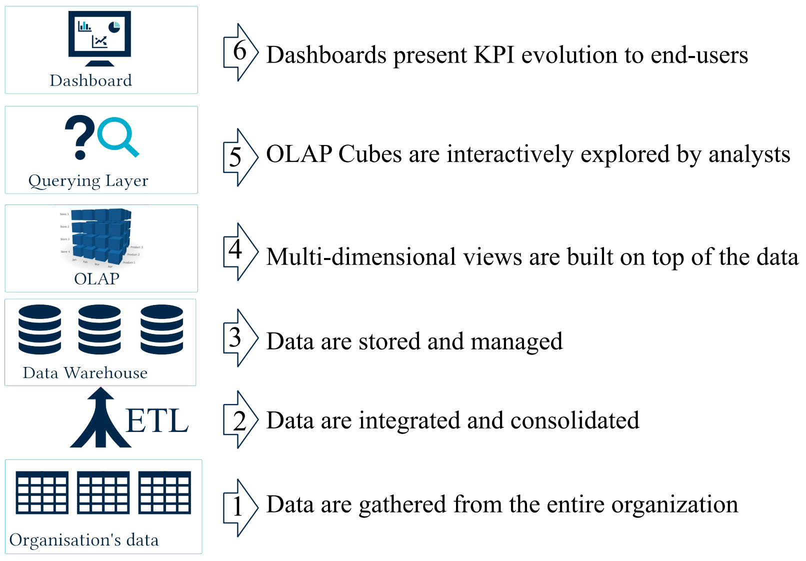

Dashboard
Querying Layer
OLAP
Data Warehouse
ETL
Organisation's data
6 Dashboards present KPI evolution to end-users
5 OLAP Cubes are interactively explored by analysts
4 Multi-dimensional views are built on top of the data
3 Data are stored and managed
2 Data are integrated and consolidated
1 Data are gathered from the entire organization

Fig. 1.2: Traditional Warehousing Process

Definition 1.1. [***Measure***] *A measure M is defined as a pair $\langle name, \phi \rangle$. The measure is computed by applying the aggregation function ϕ on a set of values.*

A measure has two components: a numerical property of a fact, e.g., the sales price or profit, and an aggregation function (such as sum and average) that can be used to combine several measure values into one. In a multidimensional database, measures generally represent the properties of the chosen facts that the users want to study from different perspectives.

Dimension Dimensions enable the multi-perspectives analysis of the factual data, while dimension hierarchies enable the multi-level analysis. Dimensions are used for two purposes: the selection of data and the grouping of data at a desired level of detail.

Definition 1.2. [***Dimension***] *A dimension D is defined by the tuple $\langle name, \Lambda, \mathcal{R} \rangle$, where $\Lambda = \{\lambda_1, ..., \lambda_n\}$ is the set of the dimension levels. Two specific levels are present for each dimension hierarchy, (1) λ_1 the lowest level called the* base level*, and (2) λ_n the highest level called* All *or Apex. $\mathcal{R}$ is a partial order on the elements of Λ and describes a directed acyclic graph (DAG) defining the hierarchy and the aggregation direction between the dimension's levels λ_i.*

Definition 1.3. [***Multidimensional Schema***] *A multidimensional schema is defined as $\mathcal{S} = \langle name, \mathcal{D}, \mathcal{M} \rangle$. It organizes the data around the concept of fact, where* *each fact consists of a set of measures $\mathcal{M} = \{M_1, ..., M_n\}$ and includes identifiers of the multidimensional space generated by the dimensions $\mathcal{D} = \{D_1, ..., D_p\}$.*

Multiple schemas were proposed in the literature to represent multidimensional data, such as star, snowflake, and constellation schemas. For the sake of simplicity, we focus on multidimensional schemas with a single fact.

Data Cube The cube metaphor is widely accepted as the underlying logical construct for conventional multidimensional models [56]. A data cube refers to the multidimensional space created by selecting a level from each dimension of the multidimensional schema and placing the measures in it. A cube is composed of a set of cuboids, where each cuboid is obtained by aggregating the original data at a given hierarchical level on each dimension. A **lattice** is used to represent all the possible aggregations of the cube. Therefore, each point of the lattice represents a data aggregation, that is, a cuboid of the cube. Given n dimensional attributes in the fact table, the cube contains 2^n cuboids. Cuboid aggregations differ from arbitrary aggregations in that they have to follow the dimension hierarchy schema when performed. A cube instantiates a multidimensional schema, where each cell of a cuboid lies at the **intersection** of all the dimensions forming the base. Cells are loaded with uniquely identifiable measures and aggregated with their corresponding compatible aggregation function. The measures are used as the basic data unit manipulated by multidimensional queries.

Given a movie dataset, Figure 1.3 shows the star schema and the lattice of a three-dimensional cube for the rating of movies. The measure is the average score of a movie computed by the average function, and the dimensions are the country of that movie, the year on which the movie received a given score and the scoring website. Figure 1.4 depicts an instance of the data cube.

Given a data cube, a set of general rules should be respected to ensure the consistency of the cube data through the different states and transformations. We refer to these rules as the multidimensional integrity constraint. Specifically, they are used to ensure the correct placement of the measures in the multidimensional space and guarantee the summarizability of the examined measures through the different aggregation levels. Summarizability is guaranteed if the three conditions of disjointness, completeness, and compatibility are guaranteed [2, 80].

Online Analytical Processing (OLAP)

Operations running on the operational databases are called on-line transactional processing (OLTP) and rely heavily on transactions to support an update-intensive workload while remaining compliant to the ACID (atomic,

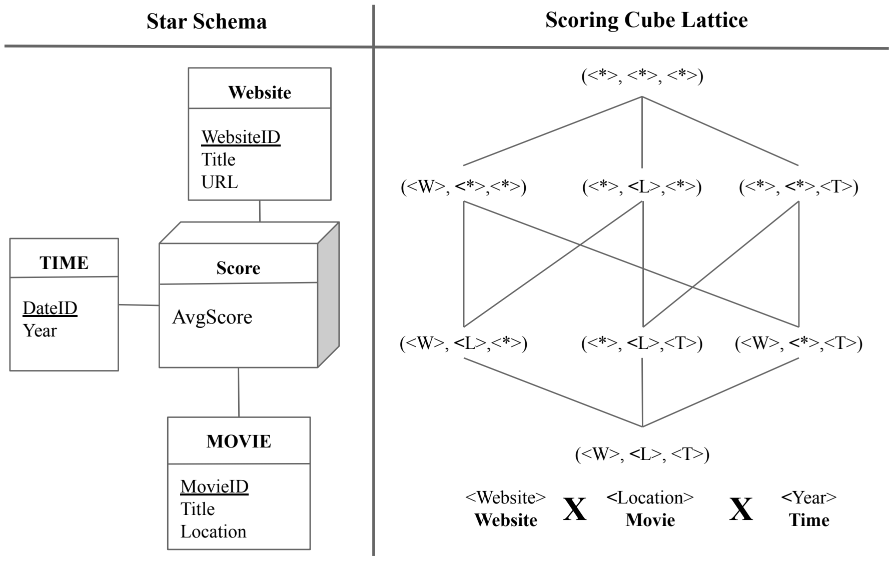


Fig. 1.3: Schema and Lattice of Movie Scoring Cube ($*$ represents the All level)

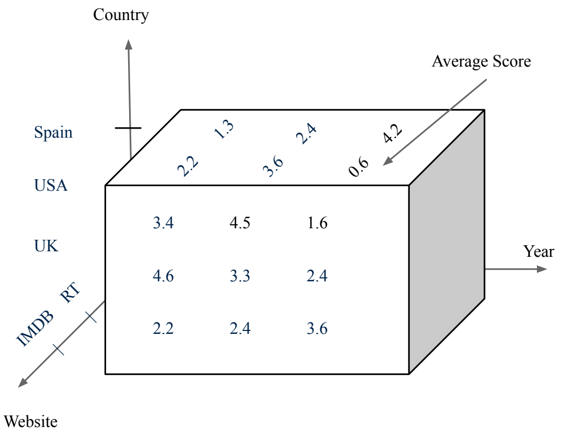


Fig. 1.4: Three-dimensional Movie Rating Cuboid

consistent, isolated, durable) properties. The goal is to avoid data redundancy and to provide a normalized schema guaranteeing consistent, up-to-date databases. Data warehouses are designed with the purpose to answer large read-intensive queries. Since such workload might overlap with the locks imposed by ongoing transactions, the data warehouse emphasizes the separation of transactional processing from analytical processing referred to as on-line analytical processing (OLAP) [36]. Techniques such as materialized views, special purpose indices, data redundancy tolerance, and denormalized schemas are applied to fit and optimize analytical workloads.

OLAP analysis follows a fact/dimension dichotomy to interactively study a business fact from multiple perspectives, called dimensions, at different aggregation levels, called dimension hierarchies. OLAP cubes embed aggregated data denoted as measures. The measures are the metrics for the analysis. Cubes are placed into the so-called multidimensional space, where dimensions are the factors influencing the values of the measures. In a relational setting, a cube for describing the sales of products would have the selling price as the measure, while time, location, and product would be the dimensions.

OLAP querying is based on a multidimensional algebra that guides navigation, querying, and analysis of data. The OLAP algebra provides a reference set of operators to handle the multidimensional data [111]. The goal is to support analysis-oriented design methodologies, better and accurate indexing techniques, and to facilitate query optimization. The core OLAP operations are (1) Selection (slice and dice), to choose a subset of a dimensional range of interest from the multidimensional space, (2) Projection, to selects a subset of measures of interest, (3) Roll-up, to summarizes data at a higher level by grouping cells based on a dimension hierarchy, (4) Drill across, to change the subject of analysis while keeping the same multidimensional space, and (6) Set Operations, namely Union, Difference and Intersection of cubes defined on the same multidimensional space and having the same set of measures. Figure 1.5 shows the transformation executed by OLAP operations on the data cubes.

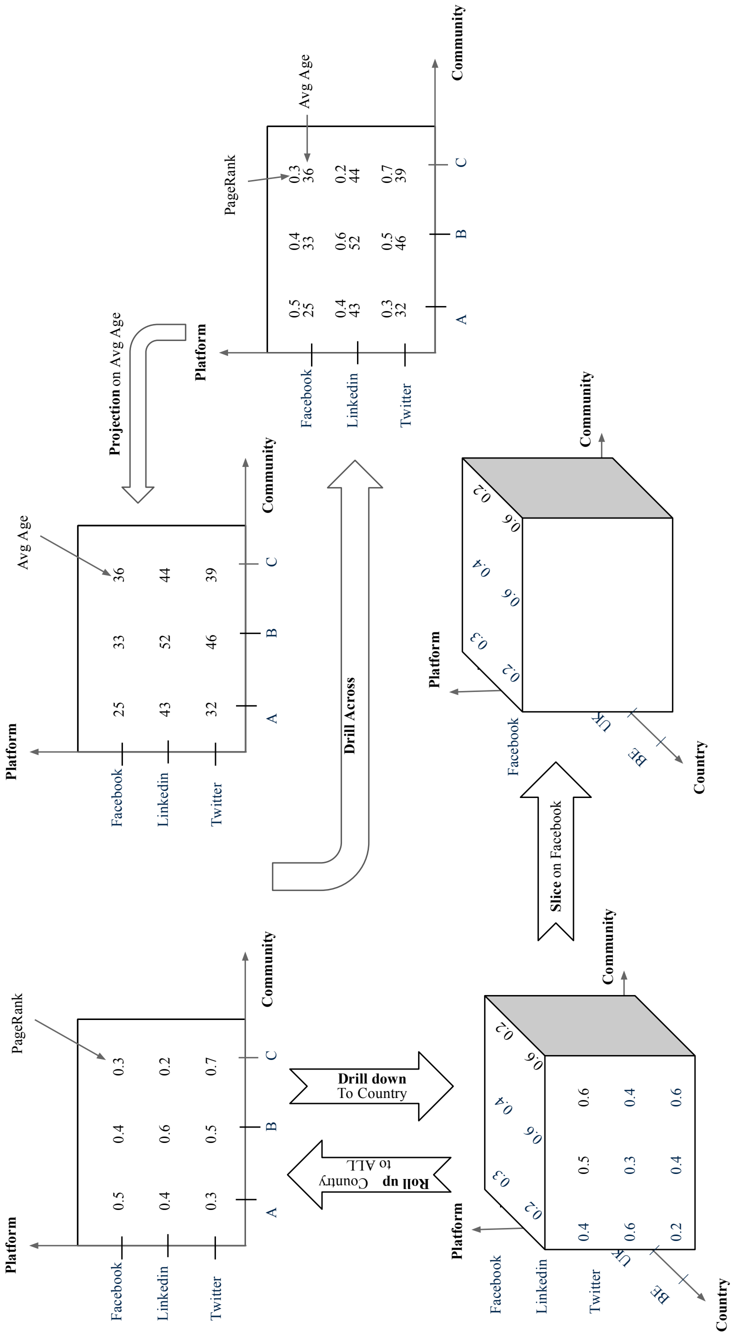


Fig. 1.5: OLAP Operations

2.2 Graph Modeling and Management

For several decades, the relational model was considered as the default choice for data modeling and management. However, the relational systems were pushed to their limits as the one-size-fits-all data management solutions and fell short of meeting the diverse requirements of emerging big data applications. Indeed, Big Data refers to data generated at unpredictable speed, in large volumes, from heterogeneous sources, and in different formats [12]. Big data applications came with challenging requirements such as high scalability and low latency. The variety of big data and its applications led to the development of new models and processing paradigms that meet those requirements. Data variety is considered as one of the most critical challenges in big data nowadays, and efficiently supporting a variety of data sources and models is considered to be the main factor of success for data-driven organizations [15]. The wide adoption of the emerging NoSQL solutions by industry, while not yet as mature as the relational model, proved the need to push databases into fields beyond the traditional business applications [20, 24].

Of particular relevance to this thesis is the study of graph data management and processing. Graphs provide an intuitive and simple knowledge representation system that naturally fits highly interconnected domains and applications. Graph analysis could be used to directly target and extract the information hidden within the graph, and reveal valuable insights from both the network structure and the data embedded within. Besides, graphs meet the requirements to be the perfect canonical data model for data integration systems [47] given the fact that (1) they can deal with heterogeneity, (2) they are semantically richer or at least as rich as any other model, (3) they allow creating multiple views from the same source, (4) they offer the capability to associate data and metadata, (4) and are extremely flexible to compose new graphs, that is, given two graphs, with one single edge a new graph could be directly created without affecting the existing ones. Graphs are, therefore, suitable to deal with the variety challenge better than any other data model. To address the need for incorporating graph analytics into business applications, many frameworks were developed for the management, processing, and analysis of graph data. In what follows, we look into these topics, and we start with the two principal aspects of the graph management: (1) graph modeling, which provides the theoretical abstraction for intuitively representing and analyzing connected data, and (2) graph databases, which are recognized as the central component for the native storage and efficient querying of graph data.

Note that through the thesis we might refer to a graph as a network, to a node as a vertex, and to an edge as a relationship, these words are often used interchangeably in the literature.

Graph Database Modeling

A database system is built on the fundamental abstraction of a database model, which provides the theoretical foundations for data management. A database model, as defined by Codd in [34], consists of a set of (1) data structures used to represent the data, (2) integrity constraints, to ensure the database consistency at its different states, and (3) manipulation operators to query and transform the data. In-line with Codd's definition, a graph database model is a model where the data structures are represented with graphs, the required integrity constraints are defined over these graph structures, and graph operators are defined to manipulate the graph. A plethora of graph models was developed to accommodate the various graph domains and applications [9, 10]. For example, data could be modeled as a simple graph or a hypergraph (i.e., a single edge connects an arbitrary number of nodes). A graph could be undirected or directed (i.e., nodes are connected with oriented edges). It could be homogeneous or heterogeneous (i.e., multiple types of nodes and edges with arbitrary properties exist within the same graph). Nodes and edges could be labeled to describe their types and weighted or attributed to denote their specific properties. A graph might be dense, where the number of edges is close to the square number of vertices, otherwise, it is called sparse. Concerning the schema, the graph could be structured or unstructured (i.e., it does not conform to a rigid predefined schema, and elements of the same class might have different attributes and data types). The graph could be considered as a single large graph, where the analysis is applied on domains such as social and bibliographic networks. In this case, the querying is focused on pattern matching, reachability, and shortest path. Otherwise, the data is considered as a collection of graphs where the domain is usually related to bioinformatics and chemical networks, with the querying revolving around subgraph and supergraph retrieval [117]. The topic of graph database modeling witnessed an increasing interest from the database community. For a deeper dive into the topic, we refer the reader to the foundational surveys of Angles and Gutierrez [6], and the recent survey by Besta et al. [20].
Currently, the two widely accepted graph models are property graphs and RDF graphs:

Property Graphs Property graphs describe a directed, labeled, and attributed multi-graph. They were first introduced by Rodriguez and Neubauer in [109], then formalized by Angles in [7]. In a property graph, each real-world entity is represented by a node and related to other entities with edges. Nodes and edges are labeled and have a set of key-value attributes. A label denotes the "type" of the node or edge (i.e., the class to which it belongs), while the attributes represent its properties and identifiers. Property graphs were in-

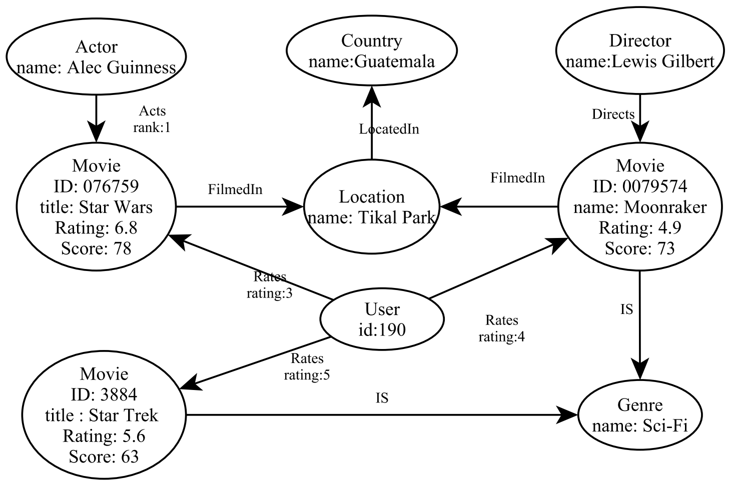


Fig. 1.6: Movie Graph

troduced in the database community to store schema-less data (due to their flexibility to absorb any semantics and attach data with metadata). Multiple query languages were proposed to enable the querying of property graphs [9], such as Cypher [46], GCore [8], PGQL [129] and Gremlin [108]. Cypher is an SQL-like declarative language, that uses isomorphism-based no-repeated-edges bag semantics. It was introduced by Neo4j and is centered around pattern matching enriched by built-in algorithmic libraries. Gremlin is a graph traversal language, built using Groovy, introduced by Apache TinkerPop3, that uses the homomorphism-based bag semantics. Given the diversity of the property graph query languages, a recent initiative is led for defining GQL [1], an ISO project to design a standard query language for property graphs by combining the best of the main property graph query languages (Cypher, G-CORE, and PGQL). Integrity constraints for property graphs is a less explored topic [7, 100], and focus mainly on schema-instance consistency, although other constraints such as attribute uniqueness or cardinality constraints were discussed. Figure 1.6 shows an instance of a movie graph, modeled using property graphs.

RDF Graphs RDF refers to the Resource Description Framework, a W3C recommendation for representing web resources. The basic RDF block is the triple, a binary relationship between a subject and an object; i.e., $\langle subject, predicate, object \rangle$. A set of RDF triples forms an RDF graph. In an RDF graph, subjects and objects are represented with nodes, while the predicates are represented with edges. The RDF resources could be one of three types: subjects, objects, and predicates could be IRI (Uniform Resource

Identifiers), subjects and objects could be blank nodes (i.e., resources without an IRI), and only objects could be literals (i.e., a constant value such a string or an integer). RDF is the basic formalism behind knowledge graphs, used to describe and link resources. RDF Schema (RDFS), a W3C recommendation, was introduced to express basic constraints on RDF triples. In the same line, the Ontology Web Language (OWL) allows expressing richer constraints and semantics. As OWL is serialized on top of RDFS, it results in a graph too. Importantly, it is also usual to refer to knowledge graphs (i.e., RDF(S) or OWL) as ontologies. This is because they can be translated to a fragment of First-Order Logic (FOL), typically, within the family of Description Logics (e.g., in the case of OWL), and benefit from generic reasoning algorithms in that field. Finally, the W3C recommendation to query knowledge graphs is the SPARQL Protocol and RDF Query Language (SPARQL). Relevantly, SPARQL enables the activation of generic reasoning capabilities when querying knowledge graphs to infer non-explicit knowledge, and SPARQL applies homomorphism when performing pattern matching. Knowledge graphs were born within the semantic web stack and therefore initially thought for enabling interoperability and reasoning on semantic data. They are tightly related to the knowledge representation community and therefore thought to represent generic knowledge rather than data as for databases. For this reason, knowledge graph databases are referred to as triple-stores rather than graph databases. One key aspect of property graphs is that they provide means (i.e., URI) to universally identify graph vertices and edges from external sources. This facilitates the linking and sharing of data and metadata. However, unlike property graphs and traditional graph databases, which are optimized for graph traversal, they are primarily optimized for handling knowledge. Another difference is that in property graphs properties could be directly added to edges as well as vertices. However, RDF* has been recently proposed, to extend RDF with properties. This boundary is increasingly fuzzier. In essence, knowledge graphs are also graphs and can benefit from the traditional graph algebras presented in the database field, and vice versa. Bit by bit, both worlds are getting closer.

Graph Management

Two main approaches are widely used for graph data management at the logical/physical level, regardless of the adopted model. The first approach leverages alternative models, mainly the relational model, and represents graphs as a set of tables, i.e., node tables and edge tables [61]. The latter advocates the use of native graph data models and database engines [3]. These approaches are discussed next:

Relational-based Databases This approach benefits from the well-established relational model features, and enables smooth integration with a wide range of relational platforms, and in some cases superior query processing performance [61]. The relational model was designed to handle structured data such as records and transactions. Tables are the fundamental construct of the relational model. A table consists of rows and columns, where each row represents a single data entity, and each column is an attribute. Relationships between tables are implemented using primary keys, which are particular columns that uniquely identify a single entity. A graph could be modeled as a collection of tables of nodes and edges. Each node is uniquely identified by its primary key, and edges are attached to their source and destination nodes using their primary keys as foreign keys. However, the table abstraction does not naturally support the modeling and exploration of graph relations and patterns. The relational query languages and processing engines are optimized to perform table scans instead of traversals. Graph traversal, simulated using expensive join operations, incurs heavy workloads, especially for highly interconnected data. Indeed, given the large cost of multi-joins, the deeper the traversal is, the more the performance degrades. Moreover, the SQL is not suited to express and handle the topology of the graph with queries such as pattern matching, neighborhood, or path retrieval. Mapping graph data to a relational representation raises the impedance mismatch problem at the modeling and querying levels. Due to the fundamental difference between the two models, transforming graph data to fit the relational model is a manual and complicated process, with a high risk of information loss. Besides, in a relational system, the schema needs to be known and fixed in-advance as changes to the schema or integration of different schemas are often costly and manual operations. In summary, the relational model and its implementations fall short of meeting the requirements for (1) intuitive graph data modeling, (2) topology-aware graph querying (such as path retrieval or graph matching), and (3) traversal-optimized performances.

Native Graph Databases In recent years, the trend in developing graph data management systems has shifted to the development of native, relationship-oriented graph databases [6, 10]. Graph databases introduce optimizations for the storage and querying of graph data by using specific graph structures and supporting the querying with a set of special operators, algorithms, and indexing techniques. Most native graph databases implement the property graph model or a variation of it. The impedance mismatch is resolved since relationships are first-class citizens, and the data is represented as it is perceived without the need to map it to intermediate representations. The data model is more straightforward to design, and the queries are more intuitive to formulate [61]. From a performance perspective, graph databases are optimized for graph traversals. Indeed, since graphs are stored using adjacency lists, the traversal between neighbors could be implemented as a simple memory lookup using a pointer attached to each node. The cost of traversing an edge is constant, and the overall cost of arbitrary navigation through the graph is much lower than the equivalent joins in relational tables. Nevertheless, complex operations such as pattern matching still yield prohibitive computational complexities if not bounded. Yet, its cost is not worsened by the added need to consider joins. Subsequent implementation aspects such as graph query processing, indexing, and storage which are specifically developed and tuned for graph workloads lead to better performances, especially for queries requiring multiple joins, or containing cycles and other complex structural patterns. However, they perform worse than the relational-based engines for analytical queries that perform scans over the whole graph and apply group by and aggregation operations [32]. In the software market, established BI vendors are aware of the potential of native graph solutions and have already developed many graph databases such as Neo4j, Oracle Spatial and Graph, Microsoft GraphEngine, IBM Graph, and Amazon Neptune.

Multi-modal Databases This is a hybrid approach that supports a graph abstraction and querying layer on top of non-graph stores such as column-store (DataStax Enterprise Graph) and document store (OrientDB, ArangoDB).

2.3 Graph Analysis and Processing

Graph analytics motivated many research initiatives for the exploration and analysis and visualization of the graph structure and content [69]. A plethora of graph analysis techniques was proposed in the literature to reveal interesting properties about the graph topology and the connectivity between graph elements [119]. A rich repertoire of algorithms was developed to efficiently solve theoretical questions such as reachability between nodes, or address NP-complete problems such as subgraph isomorphism or distributed graph partitioning. These algorithms are later used to answer a multitude of emerging decision-making problems, that are represented using graph models and solved using graph algorithms.
The core analysis operations of graphs are (1) graph traversal to assess reachability (e.g., find the shortest paths, and retrieve the neighborhood), (2) metrics computation of local (e.g., centrality), and global properties (e.g., diameter), and (3) graph pattern matching [9]. Most of these operations are supported by graph database engines. More advanced graph analytics are enabled using graph mining and distributed graph processing.

Graph Querying This field has attracted a lot of research that focuses mainly on exploring the structural properties of non or loosely attributed graphs [9, 13, 22]. A set of graph measures were developed to quantify various properties [94]. A wide repertoire of algorithms was designed to efficiently answer queries such as the shortest path, or address NP-complete problems such as subgraph isomorphism [44]. Common graph analysis techniques include:

- Path traversal: To identify all the connections between a pair of entities, used in revealing complex connections, and understanding risks and exposure.
- Centrality computation: To identify the most central entities in a network, used in detecting the influencers on a community or the critical components of an infrastructure.
- Community detection: To identify clusters or communities, which is of great importance to understanding behaviors and issues in sociology and biology.
- Subgraph matching: To search for a pattern of relationships, useful for validating hypotheses and searching for abnormal situations, such as collaborative fraud.

Graph Mining Data mining refers to the process of discovering patterns or models for data [30, 57]. In contrast to querying that retrieves known patterns, mining enables the discovery of previously unknown, implicit information and knowledge embedded within a dataset. Traditionally, data mining techniques process data as a collection of independent instances (i.e. observations). However, the recent emergence of graph structure as a rich data model involves a paradigm shift on how data mining algorithms can be applied. Graph mining algorithms provide a new way of extracting and discovering latent insight from graphs by leveraging the relationships between entities [37, 114]. However, graph mining algorithms face three main challenges [3]: (1) adapting the mining algorithms to make them graph-aware, (2) redesigning the algorithms to be implemented by those new high-performance techniques, and (3) storing and exploiting multiple but related graphs that serve for the same business purpose as in the graph warehouse [120].
A plethora of graph mining techniques was proposed in the literature such as graph clustering, frequent subgraph mining, proximity pattern mining, and link prediction [67]. These techniques are relevant in the BI context as they reveal interesting properties about the topology and the connectivity between business entities.

The historical and integrated view provided by the data warehouse makes it a suitable backbone for offering a variety of analysis scenarios. In the graph warehouse context, graph mining could be combined with OLAP to offer more capabilities both during the phase of the design and also the analysis of the graph warehouse data, in a much similar way to the traditional online analytical mining (OLAM) [57]. During the design phase, graph mining algorithms could be used to enrich the OLAP cubes with new types of topological dimensions and measures (e.g., PageRank, community). During the analysis, graph mining could assist the analyst in complex tasks such as building summarized business-oriented views of the graph, providing new perspectives to analyze the graph, or discovering interesting or anomalous patterns within the large graph cube space. In the context of outlier detection, graphs provide an elegant framework to predict and describe outliers. For example, in the context of graph cubes mining, [41] developed a measure of interestingness of patterns in a graph cube, while [21] proposed an entropy-based filter to detect interesting associations between attributed nodes in a graph cube.

Graph Processing To deal with large historical graphs, which is the case in data warehouses, graph BI systems need to integrate large-scale graph processing frameworks. Graph processing frameworks natively support the graph topology and offer graph programming models and abstractions to easily implement a multitude of graph algorithms [71, 91]. These frameworks have the capabilities to efficiently perform large scale, ad-hoc, and distributed computations over large graph data that exceed a single machine capacity. They offer features such as automatic graph partitioning, load balancing, network and disk transfer optimization, and fail-over of the processing tasks.
However, distributed graph processing poses additional challenges to centralized or traditional parallel data processing in that: (1) the graph structure is irregular, which poses challenges to the graph data partitioning and limits parallelism, (2) computation is driven by the structure, which causes a poor memory locality and data transfer issues, and (3) algorithms traverse the partitioned graph in an exploratory way, and are iterative by nature, which is I/O intensive [89, 116]. To tackle these challenges and enable efficient large-scale graph analytics, different processing paradigms were introduced [14]:

- Hadoop Family frameworks: MapReduce denotes a programming model for large-scale data processing. Hadoop is an open-source framework that supports data-intensive distributed applications, and clones Google's MapReduce framework. It is designed to process large amounts of unstructured and complex data and runs on shared-nothing architectures. MapReduce frameworks are useful for content-based aggregation of graphs (e.g., graph cube aggregation), but they are not efficient for graph-specific computations [42].

- Synchronous frameworks: Pregel [90], and its open-source implementation Apache Giraph, are distributed fault-tolerant graph processing frameworks designed to execute vertex-centric graph algorithms following the Bulk Synchronous Parallel processing (BSP) paradigm. BSP is a shared-nothing processing paradigm for parallel algorithms' execution. The computation is done as a series of super-steps over a set of processing units, each having its local memory. Each super-step consists of three phases, first (1) each processing unit performs concurrently and locally its computations, then (2) data is exchanged between the different processes, finally (3) when a process finishes the computation and communication, it reaches the synchronization barrier and it waits for the rest of the processes to finish before proceeding to the next super-step. The advantage of this paradigm is that it ensures a deadlock-free computation. However, the downside is the execution time, where the system has to wait for the slowest machines to finish before proceeding.

- Asynchronous frameworks: In contrast to the synchronous shared-nothing processing frameworks, GraphLab [87] and PowerGraph [54] are asynchronous and follows the Gather-Apply-Scatter computational model, with shared memory abstraction. These frameworks might provide better performances, but incur more complexity and higher scheduling and consistency costs.

- Hybrid Systems: These frameworks enable a mixed workload of graph-parallel and data-parallel processing. GraphX [55] is an Apache Spark [141] library developed for graph processing. It is a fault-tolerant, distributed, in-memory graph processing framework that adopts the bulk-synchronous model and is built on top of the Resilient Distributed Dataset abstraction. GraphX provides a set of primitive operators to load and interactively query the graph data. GRADOOP is a distributed framework for graph management and querying [70]. It introduces a new graph model that extends property graphs, supports Cypher queries, and the queries are processed using Apache Flink [26].

2.4 Applications of Graphs

Large complex graphs have emerged in a multitude of business domains [118]. This section reviews some of the main applications of graph analytics.

Financial Graphs The financial sector is one of the most promising sectors to benefit from the features offered by graph analysis. Several types of fraud could be detected and prevented in auction and transaction networks using graph analytics [4], including:

- Transactions graph [130]: Frauds are carried out by networks of intermediaries that mimic the legal behavior when carrying out illegal operations. To counter this type of frauds, bank transaction data could be used to build a transaction network. Each node represents a client, and each edge represents a transaction. As fraudsters tend to collaborate to orchestrate complex fraud at large scale, the probability that a customer is involved in a fraud depends on his entourage, i.e., his neighborhood in the transaction graph. Graph analytics could be used to define and retrieve complex fraud patterns, or to score customers by fraud exposure. Thereby, preventing frauds such as the opening of a bank account with false identities, the use of lost or stolen or non-existent credit cards, transfer money to illegal destinations, or exceeding legal constraints
- Invoice graph [72]: Companies are linked together by resources shared (goods, addresses, telephones, personal, etc.). Some of these companies do not contribute to the economy, do not pay their taxes, and declare bankruptcy. By analyzing the resource and invoicing networks, one can find the patterns suspicious that characterize this behavior and can detect high-risk companies.

E-commerce Graphs A set of interesting graphs are identified in the rapidly evolving e-commerce sector. Here we discuss two types of relevant graphs along with their added value:

- Purchase graph [79, 139]: This is a bipartite graph, with two types of nodes (representing products and customers). A link is established between a product and a customer if the latter has bought this product. From this graph, two more graphs could be derived: (1) client similarity graph (two clients are related if they bought similar products), and (2) similarity graph of the products (two products are related if they were bought one the same customer). These three graphs allow establishing several analysis scenarios such as (1) customer profiling by detecting customer groups, and the VIPs within each group, (2) product segmentation by detecting products representative of each segment, and (3) targeted marketing personalized to the customer's profile and tailored by current product trends.
- Delivery graph: These graphs record the delivery paths of products, returns, and any movement of stock between different warehouses. The analysis of this graph is focused on the analysis of delivery paths by searching especially the shortest or least expensive paths. This graph makes it possible, for example, to perform analysis scenarios such as (1) the optimization of the routing path and faster delivery times, and thus improving customer satisfaction and minimizing returns, (2) the

improvement of stock locality taking into account trends to minimize stock movements between transport platforms and warehouses, and (3) fight against frauds, by identifying the geographical areas that register the most of the package losses or unjustified return.

Telecommunication Graphs The telecommunications sector is particularly suitable for graphs. For example, the following graphs could be built:

- Customer and Call Detail Record graphs [40]: the integration of these two graphs provides a better understanding of the call routines and interactions between customers. This information can then be used to target marketing campaigns, prevent and manage the customer churn, and detect fraudsters. In the case of the churn, the operator could use graph-based techniques that aim to identify subscribers who are likely to churn and encourage them to stay, and especially minimize the losses caused by the departure of a major client, because it can cause the departure of its connections by leaving.
- Network infrastructure graph [121]: this graph can be used to model the different infrastructure components (e.g., routers, antennas, switch, etc.). By detecting the central nodes and links, graph algorithms make it possible to better identify the critical network points. This helps to reduce network congestion and increases performance by optimizing information exchange.

Source-code Graphs Software development projects can quickly grow in volume and complexity (millions of lines of code). Graphs are relevant in this field as they allow an intuitive representation of the source code and its evolution through the different versions. Graph analytics could help understand the logic encoded in programs (e.g., by analyzing the transitive closures and hierarchies of call methods), and the removal of redundancies in the code (e.g., by mining the source code to discover the frequent patterns). The main source code representations using graphs are [138]:

- Abstract Syntax Tree (AST): This is a tree whose internal nodes are marked by operators and whose leaves (or external nodes) represent the operands of these operators.
- Control Flow Graph (CFG): A control flow graph is a representation under the graph form of all the paths that can be followed by a program during its execution.
- Program Dependence Graph (PDG): The dependency graph represents the data and control dependencies of a program which allows the detec-

tion of potential conflicts and undesirable dependencies between software libraries.

Graphs applications at the industrial partner

At EURA NOVA, graphs are considered as an important research topic and a promising technology that often brings a competitive advantage. Many graph-related research directions were explored such as graph-based image retrieval, graph databases design and benchmarking, knowledge graphs, etc. This Ph.D. was proposed to explore the promising integration of graphs within decision-making systems, and particularly data warehouses. The challenges discussed above are encountered by EURANOVA's customers, and we found graphs to be a natural fit to model and solve many of their challenges. Here we briefly discuss three use-cases:

- **Fraud Detection and Prevention:** Graph OLAP techniques can also be used to examine potentially fraudulent behavior across a variety of domains such as Telco and financial services. For example, we used graphs to represent the network of customers and quantify their exposure to fraud through various dimensions such as proximity, location, and age. This enabled OLAP queries such as (1) retrieving the customers most and least exposed to fraud by year, city, and age, and (2) calculating the average exposure to fraud for a customer segment from a given location at a specific period, to assess the evolution of the exposure to fraud after prevention measures have been taken.

- **Customer** 360**:** Companies thrive to understand their customers and adjust their offerings to meet their expectations. This use-case often requires the integration and display of heterogeneous entities and their relationships. Graphs are often a natural fit when it comes to building a 360 platform, thanks to their intuitiveness and flexibility.

- **Commerce Organization:** Optimizing the distribution of shops in Brussels is a challenging task given the situation in the city center which is regularly affected by various events (mobility plans, alert levels, various events, etc.). EURA NOVA partnered with a governmental agency to draw insights that would be useful for the revitalization of Brussels businesses. In this project, we represented the data about shops as a graph that described the different commercial activities of Brussels, as well as the profiles of the customers frequenting these businesses. Graphs offered a new multidimensional perspective on the data, which allowed a joint analysis of the commerce distribution and activities, and their interconnection with the commercial pedestrian flow in Brussels over different periods.

As customers are accustomed to using BI as the cornerstone for their data analytics and decision-making processes, preserving the familiar BI while changing some components to support graphs seemed the best trade-off. Therefore, we experimented with a wide range of solutions to address the integration of graph analytics within the decision-making tools. In most cases, businesses need an interactive graph dashboard, equipped with some graph derived metrics, and the capability to query the graph. These requirements could not be satisfied with a simple graph layer built on top of a relational system but needed an optimized graph storage and processing engine. In our case, this translates to a dedicated graph warehousing solution capable of efficiently storing and processing large graphs cubes. However, when trying to fulfill these requirements, we realized there were no frameworks ready to combine BI and graphs. To the best of our knowledge, none of the current data warehouse systems supports graphs. Seemingly, after checking the academic state-of-the-art, we found several frameworks but with strong assumptions (homogeneous graphs, no specific graph warehousing frameworks, not considering graph topology when rethinking the multidimensional constructs on graphs, etc.). Therefore, we formulated the following objectives in the next section.

3 Objectives and Contributions

In this thesis, we argue for the design of a new database and multidimensional graph models that reflect the rich topological information embedded in graphs and integrate it within warehousing environments. This section presents the main objectives and research questions explored in the thesis.

Objective 1: Graph Database Modeling for Analytics Graphs have been an active field of studies for decades [94]. The importance of graphs as a fundamental structure underpinning many real-world applications is no longer to be proved [3]. A plethora of frameworks were developed for the processing, management, and analysis of graph data [10, 14]. In particular, graph databases play a key role as the backbone for efficient storage, management, and querying of graph data [20]. However, current graph database models and systems are mainly designed for OLTP workloads and do not provide the required foundations for graph warehousing. Indeed, graphs and data warehousing are, in essence, contradictory from a modeling perspective. Graphs advocate for a flexible schema-less approach, while warehousing requires strict modeling to automate ETL and OLAP cube computation. Emerging models such as property graphs propose simple data structures and are loosely constrained, which makes it difficult to deterministically define and map advanced multidimensional concepts to graph structures or to

later automate tasks such as the identification of potential multidimensional concepts and extraction of graph cubes. Moreover, due to the rich properties and literature on graphs, one faces multiple options when designing a graph database model. For example, whether to work with graphs, hypergraph, or triples, whether to support directed or undirected graphs, which graph elements should be attributed or labeled, etc.
Therefore, with the rising importance of graphs in industry, we witness a growing need for new graph database models optimized for advanced analytics of large graphs, and particularly for warehousing and OLAP workloads.

To accommodate graph warehousing, a graph database model is expected to provide (1) analytical graph structures that capture rich semantics such as hierarchies, (2) a set of operators to manipulate the graph and extract graph metrics, and (3) a set of rules to enforce and preserve the integrity of the graph data through the different integration and transformation operations. We identify this need for extending data warehousing with graph analytics, and particularly the need for a graph database model capable of supporting warehousing scenarios, as a first challenge to be addressed in this thesis. We start from a thorough analysis of the current state-of-the-art to identify potential drawbacks and formulate the first open question tackled in this thesis as follows:
Research Question 1: *"how to model a graph to make it ready for warehousing and multidimensional analysis?"* More specifically:

1. What are the graph data structures oriented for graph warehousing and OLAP analytics?
2. What are the canonical algebraic operators required for the manipulation and OLAP analysis of graphs?
3. What are the integrity constraints that need to be enforced to guarantee the integrity of the graph data through the different transformations?

Contribution 1*:* We answer this first question by proposing **GRAD** (GRAph Database model), a native graph database model tailored for graph warehousing, in compliance with Codd's definition of a database model. GRAD formalizes the graph data structures, integrity constraints, and algebraic operators required from a graph modeling perspective to support graph warehousing. Therefore, GRAD defines advanced *graph structures* to capture and project the common notions present in established conceptual data modeling languages on graphs. These structures are semantically rich and self-descriptive (e.g., entity node, aggregation edges, etc.), and natively support complex semantics such as encapsulation and composition (e.g., using hypernodes) to simplify data integration and multidimensional modeling tasks. Besides, to enforce the graph data consistency, GRAD defines a set of integrity

constraints tailored to its graph data structures. Two types of constraints are defined: (1) Graph entity integrity, used to guarantee that each real-world entity is uniquely represented and identified, and (2) Semantic constraints, which are user-defined constraints that represent the assertions on the graph elements enforced by graph pattern checking. At the *operators* level, GRAD proposes a set of algebraic operators oriented for online graph querying of GRAD structures while preserving the integrity constraints. The analysis is mainly performed through traversal and matching operations, and GRAD structures are the operands and the return type of all the operators. The algebra is defined on the lines of relational algebra and inherits at least the same expressive power.
The design choices made in GRAD were taken with graph warehousing as the main use-case. For example, GRAD facilitates data integration by introducing attributes reification which simplifies data integration and historization. As illustrated through the algebraic operations, GRAD graphs could be easily composited without losing changes in node attributes. Hence, it could serve at both the ETL and the multidimensional analysis phases. In a graph setting, the ETL process should support tasks such as identification, transformation, matching, and insertion of the incoming nodes and edges in the graph data warehouse. As explained through Chapter 3, the algebra, and specifically the composition and join operations are particularly well-suited for the implementation of the transformation operations. Moreover, data quality could be enforced using integrity constraints checking mechanisms. At the multidimensional design phase, each class of hypernode in GRAD is a potential dimension. Multiplicities are implicit for specific types of edges (e.g., composition and aggregation are to-one relationships). The information inferred from relationship types or explicitly given as a constraint by the user could help in designing the OLAP dimension hierarchies and support the graph cube aggregation process.
GRAD is detailed in Chapter 3, and was published in the paper [50], which was in part based on the earlier work of [52] presented in Appendix A. The work published in [52] is subsumed by [50] and it is therefore listed in the appendix for completeness, but it is not required to follow the thesis flow.

Objective 2: Multidimensional Modeling and OLAP Analysis of Graphs
The added-value brought by graph analytics to decision-making systems motivates the need to extend current data warehouses with graph capabilities. This calls for the design of a multidimensional graph model to organize the data and re-think OLAP analysis to capture and expose graph topology and content. The initial efforts on graph warehousing laid the foundation for the multidimensional modeling and analysis of graphs [103]. The first papers introduced the notions of graph OLAP and graph cubes, formal-

ized new graph OLAP operations, and designed custom materialization techniques. However, most of the current approaches simply map relational-like multidimensional concepts to graphs, without exploiting its topology. They partially tackle the graph warehousing problem by making strong assumptions and being restricted to simple abstractions such as homogeneous or non attributed graphs. Advanced topics such as dimension hierarchies and graph-specific multidimensional concepts are not thoroughly studied. Besides, multidimensional integrity constraints on graphs, needed to guarantee the integrity of the multidimensional space, were completely overlooked by current approaches. Finally, the proposed graph cubes expose only partially the graph topology. These missing blocks in the proposed approaches leave room for improvement when it comes to designing and building a graph warehousing model aware of the topological aspects of the graphs and ready for industrial deployment. This leads to the second questions we tackle in this thesis:
Research Question 2: *"how to design a multidimensional model aware of the topological characteristic of the graph, and how to leverage these properties in OLAP analysis?"* The challenges raised here could be articulated around the following questions:

1. How to project the conceptual multi-dimensional structures on graphs to design the so-called multidimensional graph, while leveraging its topological properties. Specifically, what is a dimension, a measure, a fact, and a cube in a graph?

2. What is the advantage of using our analytics oriented graph model when designing the multidimensional graph model compared to using a property graph or the relational model?

3. What kind of novel graph cubes could be extracted from a multidimensional graph?

4. What are the algebraic operators and algorithms needed to compute and query graph cubes?

Contribution 2: We answer this second question in two phases. First, we define the multidimensional concepts for heterogeneous attributed graphs and highlight the new types of measures that could be derived. We apply this on property graphs and explore how to extract the candidate multidimensional concepts and build graph cubes. Then, we extend the multidimensional model by integrating GRAD. Indeed, GRAD explicitly supports rich types of edges such as generalization and aggregation and enables a finer organization of the content of nodes using hypernodes. We use these characteristics to support dimension hierarchies and build new types of OLAP cubes on graphs. In particular, we illustrate how this extension enables supporting

advanced concepts such as dimension hierarchies within multidimensional graphs. As a result, we show how graph modeling based on GRAD eases and facilitates multidimensional modeling.
In the second phase, we propose **TopoGraph**, an end-to-end graph data warehousing framework that extends state-of-the-art models with new types of cubes and queries combining graph-oriented and OLAP querying. TopoGraph goes beyond traditional OLAP cubes by considering the topological properties of the graph elements. And it goes beyond current graph warehousing models by proposing new types of graph-structured cubes. These cubes embed a rich repertoire of measures that could be represented with numerical values, with entire graphs, or as a combination of them. The correspondence between these graph cubes and traditional OLAP cubes is studied and the result motivates the need for native graph warehousing systems. Besides, multidimensional integrity constraints on graphs, completely overlooked by current approaches, are discussed, and we show how TopoGraph cubes guarantee them. Multidimensional integrity constraints are required to ensure the design of sound multidimensional spaces and the correct placement and summarizability of graph measures throughout the different perspectives and aggregation levels. Given the graph cubes, we define a set of operators aiming at providing answers to complex questions, that require the analysis of both the content and the topology of the graph. The algebraic OLAP operators for the proposed graph cubes were designed on the lines of the multidimensional algebra. Thus, enabling complex graph querying and OLAP analysis of the topology and content of graph cubes from different perspectives and through different aggregation levels. TopoGraph was implemented and experimentally validated with different types of real-world datasets, and used at the core of a social network analysis framework.
We present this work in Chapters 4 and 5, published in papers [49] and [51] respectively.

Objective 3: Architecture of a Graph Warehouse Data warehouses are considered as the reference central information repository for decision-making. Current data warehousing systems are based on the relational model and were developed to handle tabular structured data. However, relational systems are not well-equipped for the efficient analysis and aggregation of the topological information characterizing graph data [120]. Graph warehousing is therefore an emerging field that extends current information systems with the capability of multidimensional management and analysis of graphs [35, 82, 85]. The variety, complexity, and sheer volume of graphs in a data warehousing context pose challenges to traditional techniques for storage and analysis of graph data. Besides, graph cubes have a different structure from their relational counterpart, and OLAP operations exhibit different access and manipulation patterns. Given these specificities of graph warehousing, there is a need for unified graph warehousing architecture, designed on the lines of traditional warehousing, but optimized for graph storage and processing, aware of the specific characteristics of graphs, and leveraging its structural properties. The third question we tackle in this thesis is, therefore: ***Research Question 3:*** *"what are the main building blocks of a graph warehouse, and how could they be unified in a comprehensive architecture?"*

Contribution 3*:* We answer the third question by proposing an architecture of a graph warehousing framework augmented with machine learning capabilities. While adopting a similar template as the traditional BI and warehousing systems (i.e., it preserves the familiar analytics workflow), we extend current systems with graph-aware components that deliver graph-derived insights. The architecture illustrates how the graph data flows from the sources, through the ETL process, the graph warehouse, marts, and graph cubes to the analysis layer. At the core of the proposed architecture, we position GRAD as the recommended database model, and TopoGraph as the warehousing framework covering the graph cubes' computation and analysis. Besides, we enrich the architecture with a set of other complementary components for optimization and assistance purposes. This includes machine learning-based features such as the automated discovery of potential multidimensional schema and cubes and the extraction of interesting patterns. The integration of graph warehousing and machine learning lays the foundations for promising future research directions. Through this thesis, we implemented multiple parts of this architecture. We proposed prototypical implementations for both GRAD and TopoGraph and illustrated their usage in warehousing scenarios. Besides, through the collaboration with the industrial partner, we built a multitude of graph libraries and dashboards related to the proposed architecture. For example, we implemented a distributed algorithm for OLAP cubes computation and aggregation using the Map-Reduce model on top of Spark. And we designed an algorithm for the discovery of interesting patterns in large graph cubes. Together, these libraries provide an implementation of the architecture and a first prototype of an end-to-end graph data warehouse.
This architecture is introduced in Chapter 6 and in [48]. We explored some of the suggested improvements in master thesis collaborations and published the results in the papers [41, 42, 126].

Figure 1.7 presents a simplified version of the graph warehousing architecture, presented in more detail later in this Chapter. The components are numbered, and they related to our work as follows:

- GRAD is designed to model the data at the graph warehouse level (component 2).
- TopoGraph covers the design, computation, and querying of graph

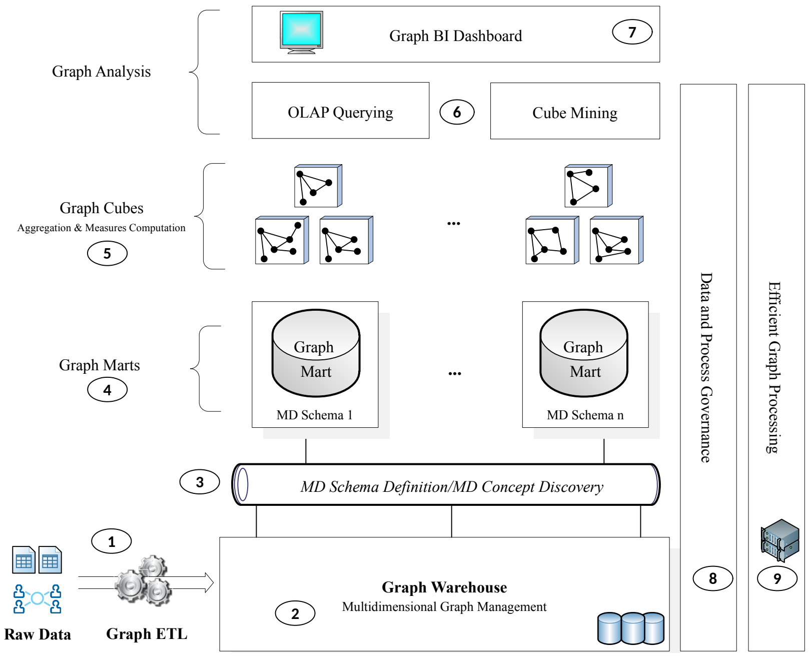


Fig. 1.7: Graph BI Framework Architecture

cubes (components 3, 4, 5, and 6).

- We built multiple graph analysis applications with custom dashboards as part of our collaboration with the industrial partner (component 7).
- The efficient computation of graph cubes and the auto-tuning of graph processing were tackled in a complementary work in papers [42, 48, 126] (component 9).

Component 1 refers to the ETL process, although we developed scripts to transform raw data into GRAD and property graphs, a systematic study of graph ETL remains out of the scope of this thesis. Component 8 refers to data and process governance, which is a metadata layer systematically tracking what happens in the warehousing system and falls as well out of the scope of the intended work.

4 Thesis Overview

This section summarizes the chapters of this Ph.D. thesis. For details on the formal definition, algorithmic framework, and experimental results we refer readers to the corresponding chapters. This thesis is organized as a collection

of individual research papers. Each chapter is self-contained and can be read in isolation. There can be some overlaps of concepts, examples, and texts in the introduction and sections of different chapters as they are formulated in relatively similar kinds of settings.
The papers included in this thesis are listed below:

1. The graph database model GRAD, presented in Chapter 3, is based on the article: *GRAD: On Graph Database Modeling* [50].

2. The design of the multidimensional model, and the graph cubes extraction techniques, presented in Chapter 4, are based on the conference paper: *A Framework for Building OLAP Cubes on Graphs* [49].

3. The advanced OLAP modeling and analysis techniques on graph cubes, presented in Chapter 5, are based on the journal article: *TopoGraph: An End-To-End Framework to Build and Analyze Graph Cubes [51].*

4. The graph BI overview and architecture presented in Chapter 6 are based on the conference paper: *Graph BI & Analytics: Current State and Future Challenges* [48].

5. The evolving graph model presented in Appendix A is based on the conference paper: *An Analytics-Aware Conceptual Model for Evolving Graphs* [52]. This paper is subsumed by [50] and it is therefore listed in the appendix for completeness, but it is not required to follow the thesis flow.

4.1 Graph Database Modeling for Analytics

This chapter addresses the need for a native and analytics-aware graph database model by designing GRAD (GRAph Database model).

Data Structures Multiple conceptual models, such as UML and EER, exist in the literature to support and standardize the representation of different domains and applications. We map the common core entities from these models to GRAD, and formally define them using the graph data structures as depicted in the meta-model of Figure 1.8. In GRAD, we consider attributed, labeled, directed multi-graphs, and extend property graphs by assigning specific semantics to graph nodes and edges. The data structures assist the designer in his task of representing and organizing the real-world entities using graphs as first-class citizens. The key data structures in GRAD are:

- Entity node: a labeled attributed node that represents the core information about a real-world graph element. The label denotes its class, and the attributes are its unique identifier.

- Attribute node: a labeled node, where the label denotes an attribute of the entity node that is not an identifier.
- Literal node: an attributed node that stores the value of its associated attribute node in a specific given context.
- Entity edge: a labeled and attributed edge that links a pair of entity nodes. It has a type that specifies the relationship which could be: Association, Generalization, Aggregation, or Composition.
- Attribute edge: represents a composition relationship between an attribute node and its parent entity node. Attribute edges do not embed attributes or labels.
- Literal edge: an attributed edge that reflects a composition relationship between a literal node (part) and its parent attribute node (composite). Attributes describe the context of the value of the literal node.
- Hypernode: a subgraph that groups an entity node v_i, the set of all the attribute nodes attached to v_i, and the set of all the literal nodes attached to the attribute nodes of v_i. In addition to all the edges linking the attribute and literal nodes related to v_i. The label of an entity node defines the class of the hypernode.

In short, GRAD brings the power of meta-modeling and modeling from traditional software engineering to graphs, as required for data warehousing. The schema part of Figure 1.8 shows how the movie graph could be modeled using GRAD, and the instance part illustrates an instance of the schema focusing on the rating subgraph related to the movie.

Integrity Constraints They are the general rules describing the consistent database states, or change of states, or both [34]. The goal of these constraints is to guarantee the compliance of the graph data with respect to given domain-specific rules. They play a fundamental role in data quality enforcement within data management systems, and especially for decision support systems. We identify two integrity constraints relevant to GRAD:

- Graph entity integrity: This constraint is used to guarantee that each real-world entity is represented by a unique hypernode. It also provides the mechanisms for nodes and edges identification through specific attributes (i.e., ID) and/or structural properties such as neighborhoods. This prevents data redundancy of data and helps to fulfill consistent updates and deletions of graph entities.
- Semantic constraints: These are user-defined constraints. The first type represents the assertions (i.e., topological and value-based constraints)

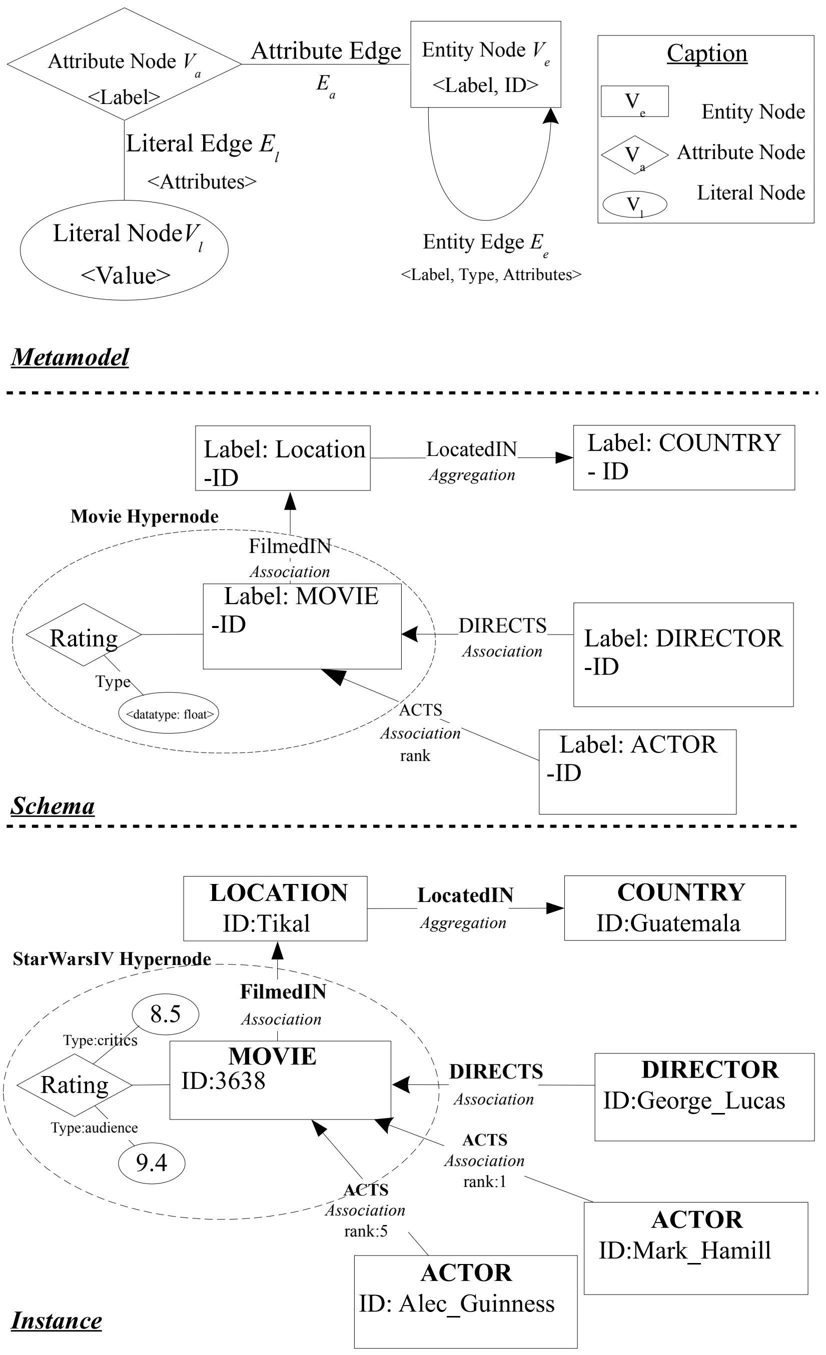


Fig. 1.8: Modeling Movie Graph with GRAD structures

defined on the graph elements. The second type focuses on cardinality checking between classes of nodes. We study two categories of such constraints: assertions and multiplicities. Assertions are predicates applied to the graph data and must always be satisfied. In GRAD, they are represented using graph patterns to specify the topological and content-based constraints chosen by the user. Multiplicities are applied between classes of entity nodes to define the number of relationships a node from a given class can have with nodes from other classes through a particular entity edge.

For example, an assertion could be defined as "all movies on the graph should have an attribute node labeled *Rating*, with the rating value *Audience* between 0 and 10". Multiplicity could be that an actor can participate in many movies but that a movie should have at least one actor, which is specified on the *ACTS* edge.

Algebra We complete GRAD with a graph algebra, consisting of a set of algebraic operators that supports its data structures while preserving the integrity constraints. The algebra extends GraphQL, a relationally complete graph algebra defined in [60]. The algebraic operators of GRAD are:

- Selection: a subgraph extraction operation based on graph pattern matching. The selection returns the subgraph (or set of subgraphs) of the data graph that satisfies the content and topological constraints specified by a given graph pattern.
- Cartesian Product: a binary operator applied to put together two collections of graphs. The input is two collections of graphs $S1$ and $S2$. The result is a set of pairs of unconnected graphs. Each graph on the output is composed of a graph from $S1$ and a graph from $S2$ respectively.
- Composition: reuses the information extracted from the input graph data to generate new graphs. This operator is used to create a new graph based on data collected from the original graph, then formatted according to a given graph template.
- Set operators (Union and Difference): A union between two graphs simply generates a new graph putting together the two input graphs without concatenation. The difference between graphs removes isomorphic subgraphs that exist in the two input graphs.
- Structural Graph Join: consists in the unification of nodes and edges based on a common join predicate.

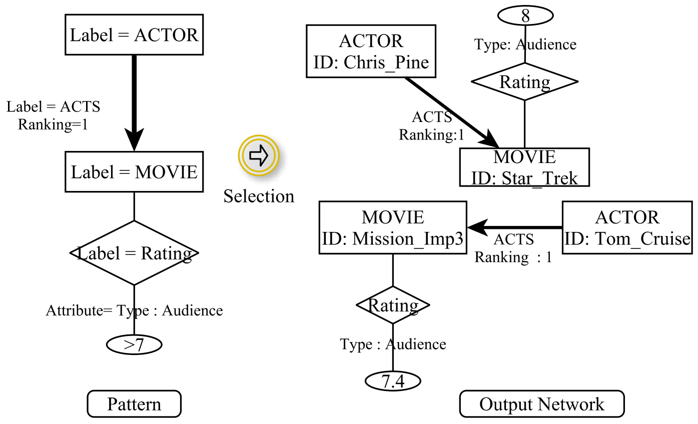


Fig. 1.9: Top movies and actors network

Example 4.1 (Top Movies Selection and Integration)
Figure 1.9 shows a selection operation to return the network of the top-rated movies by the audience (i.e., movies rated above 7), and the first actor of each of those movies. The pattern applies constraints on the labels of entity nodes ("ACTOR" and "MOVIE"), the label and attribute of their linking entity edge (the label is "ACTS", and the attribute is "Ranking = 1"), the labels of movies' attribute nodes (the label is "Rating") and the rating values (value in the literal node > 7).
Figure 1.10 illustrates a join operation used to integrate new incoming data about movies with the existing graph. Therefore, existing entities have to be merged with their corresponding incoming ones. In this example, the correspondence between the existing nodes and potential incoming nodes is performed based on the identifiers of the movie nodes. After the fusion, we can notice that the two movie nodes (with ID= 3884) are merged and an additional type of attribute nodes (Score) is attached to the existing movie as shown on the first join on Figure 1.10. Once the entity nodes and their attribute nodes are merged, another possible scenario could be the fusion of attribute nodes of the movie node. This is shown in the second join in Figure 1.10, thus adding the new captured rating values as literal nodes attached to the Rating attribute node.

4.2 Multidimensional Modeling of Graphs

Many approaches were proposed in the literature to extend current decision support systems with graphs [103]. They suggested the first foundations for

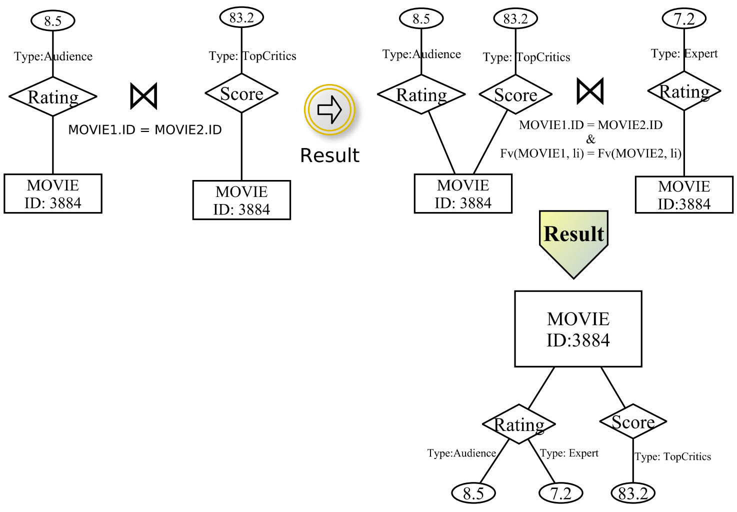


Fig. 1.10: Join on attribute and value nodes

building OLAP cubes on graphs. However, they took strong assumptions such as homogeneous graphs (i.e., graphs where all nodes are of the same type, and all edges are the same), and focused mainly on the aggregated graph as the measure of interest [135, 143]. However, real-world graphs are complex and often heterogeneous, and more types of measures and dimensions could be explored. In this chapter, we present our extension of graph cubes to heterogeneous graphs, where not all attributes could be considered as dimensions, and we examine a new class of measures to get additional insights from the graph topology. We extend the analysis capabilities on graphs by integrating GRAD, and support advanced concepts such as dimension hierarchies and build additional OLAP cubes on graphs.

Multidimensional Concepts on Graphs We first present the multidimensional concepts on property graphs, then we illustrate them with examples applied to the movie graph.

- Dimension: provides the possible perspectives for the analysis of the graph topology and content. In graphs, we distinguish two types of dimensions: (1) Node-based dimensions, which are represented by the attributes of the nodes, and (2) Edge-based dimensions, which are represented by the attribute of the edges.

- Measure: is the basic unit of data that is placed in the multidimensional space and examined through the dimensions. It is computed over a graph using a function such as the PageRank algorithm, then an aggregation function (e.g., SUM, AVG, etc.) is used to compute an aggregated value of the measure.

- Graph Cube: corresponds to a set of aggregate graphs obtained by restructuring the initial graph in all possible aggregations following the dimension. An aggregate graph $\mathcal{G}'$ of an initial graph $\mathcal{G}$ is a graph obtained by condensing a subset of the nodes and edges of $\mathcal{G}$ that satisfy the aggregation pattern.

To sum-up, a graph cube is the fundamental structure supporting the multidimensional modeling and analysis of the graph data. It consists of multiple graph cuboids, each of which is a multidimensional aggregate graph built by aggregating the original multidimensional property graph using the dimensional attributes. The lattice is used to represent and organize all the possible multidimensional aggregations of the graph. Graph cuboids relate between them when a cuboid contains an attribute with a roll-up relationship, i.e., belong to the same dimension hierarchy and are directly related. Given n dimensional attributes, the graph cube contains 2^n graph cuboids that could be aggregated following the lattice structure. We distinguish two particular graph cuboids: (1) the base graph cuboid (where the multidimensional graph is at the base level), and (2) the apex graph cuboid (where the multidimensional graph is aggregated to the top-level).

In addition, we classify graph measures, based on the type and the computation algorithm as follows:

- **Content-Based Measures:** are extracted from the attributes of graph elements. These measures are similar to traditional measures and do not capture the graph topology.
- **Graph-Specific Measures:** capture the topological properties of graphs and are obtained by applying graph algorithms. They could be classified according to the type of their output as either (1) *numerical*, or (2) *topological*, where the measure is represented using graph structures.
- **The Graph as a Measure:** the graph itself could be considered as a measure examined from different perspectives and at different aggregation levels [28].

Given a property graph and a pair of nodes from two connected but distinct classes of nodes, we explore the candidate dimensions, measures, and cubes that could be built by exploring the graph of these two classes. In particular, we identify *inter-class dimensions* as the dimensions that span across two linked classes. In this case, the candidate node-based dimensions are the attributes of the two nodes each from a class, and the candidate edge-based dimensions are a subset of the attributes of the edge relating them. Similarly, *inter-class measures* are computed by applying an aggregation function

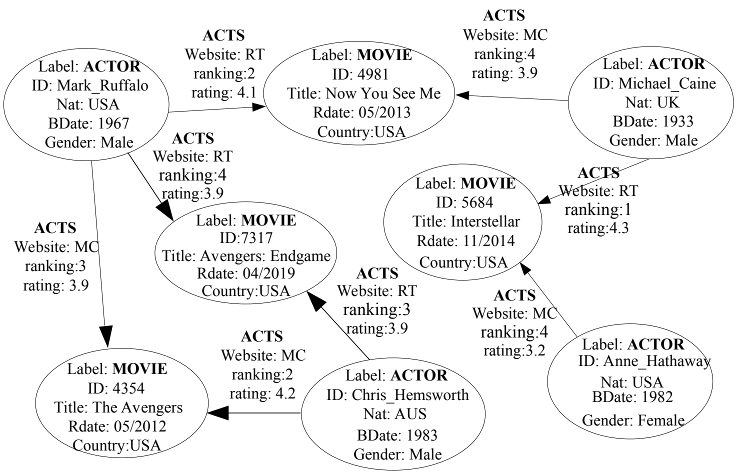


Fig. 1.11: Sample Movie Graph

on the attributes of the edges. The graph considered as a measure is obtained following the graph lattice, and the graph-specific measures are obtained by applying a graph algorithm on the multidimensional graph or one of its aggregate graphs.

Example 4.2 (Analysis of Rating and Ranking of Actors)
We take as an example and instance of the movie property graph (Figure 1.11) and a multidimensional schema of actors rating (Figure 1.12), and illustrate in Figure 1.13 how to derive graph cubes from a movie graph represented using property graphs. For example, the attribute gender and release date could be considered as a dimension, while the attributes ranking and rating are used to derive measures. A graph cube could be derived to study the ranking and rating of actors in the movie graph. Figure 1.13-(a) shows the aggregate graph (i.e., graph cuboid) where movies are grouped by release date, and actors are grouped by birth date and gender. A corresponding OLAP cube is shown in Figure 1.13-(b). The measures are *AverageRanking* and *AverageRating* of actors, which can be examined through the three dimensions left (i.e., *ReleaseDate*, *DateOfBirth*, and *Gender*). Following the graph aggregation as depicted by Figure 1.13-(e), the graph (Figure 1.13-(c)) and the cuboid (Figure 1.13-(d)) at the next aggregation level are derived.

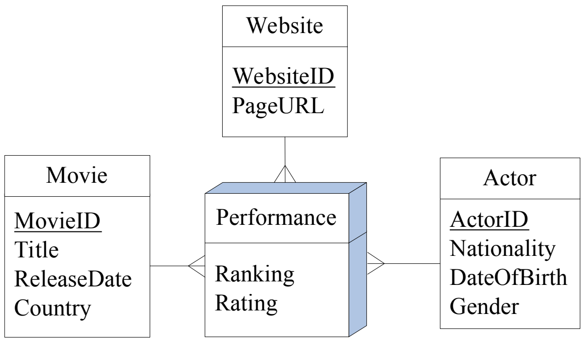


Fig. 1.12: Star Schema of Actor Rating

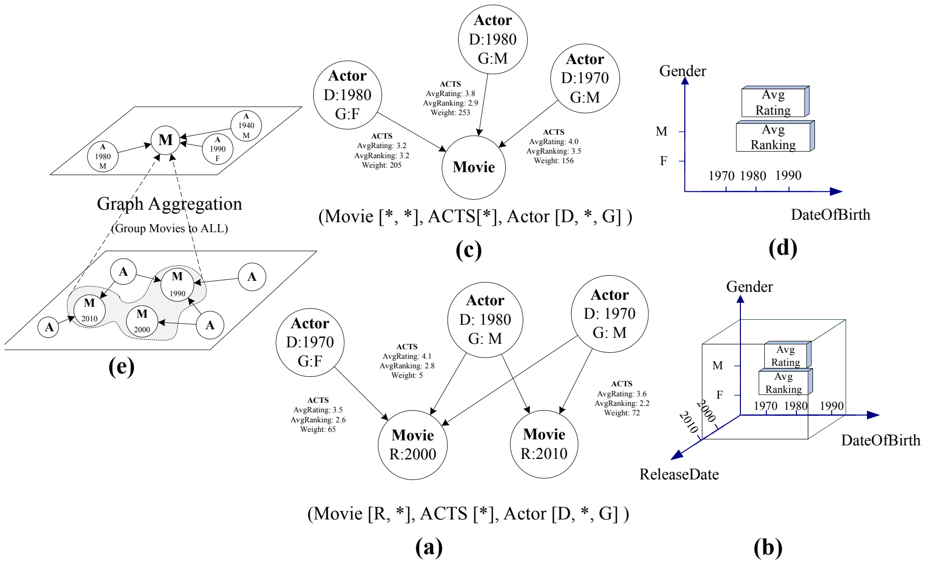

A 1980 M
M
A 1990 F
A 1940 M
Graph Aggregation
(Group Movies to ALL)
A
M 1990
A
M 2010
M 2000
A
A
(e)
Actor D:1980 G:F
Actor D:1980 G:M
Actor D:1970 G:M
ACTS AvgRating: 3.8 AvgRanking: 2.9 Weight: 253
ACTS AvgRating: 3.2 AvgRanking: 3.2 Weight: 205
ACTS AvgRating: 4.0 AvgRanking: 3.5 Weight: 156
Movie
(Movie [*, *], ACTS[*], Actor [D, *, G])
(c)
Gender
M
F
Avg Rating
Avg Ranking
1970 1980 1990
DateOfBirth
(d)
Actor D:1970 G:F
Actor D: 1980 G: M
Actor D: 1970 G: M
ACTS AvgRating: 4.1 AvgRanking: 2.8 Weight: 5
ACTS AvgRating: 3.5 AvgRanking: 2.6 Weight: 65
ACTS AvgRating: 3.6 AvgRanking: 2.2 Weight: 72
Movie R:2000
Movie R:2010
(Movie [R, *], ACTS [*], Actor [D, *, G])
(a)
Gender
M
F
Avg Rating
Avg Ranking
2000
2010
1970 1980 1990
DateOfBirth
ReleaseDate
(b)

Fig. 1.13: OLAP aggregation of the movie graph and computation of the OLAP cubes

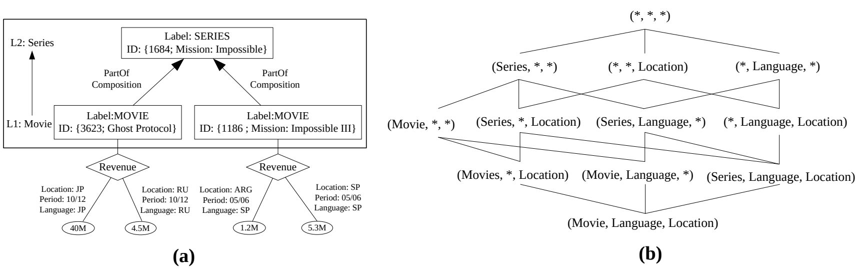


Fig. 1.14: Aggregation of revenue by language

OLAP Cubes on GRAD GRAD supports the modeling and analysis of heterogeneous, attributed, and labeled graphs, where complex attributes are supported on the nodes, and rich semantics is explicitly expressed on the edges. In the previous paragraph, we used property graphs to study the candidate multidimensional cubes between classes of nodes. Here, we explore the additional candidate dimensions, measures, and cubes that could be extracted from a single class using GRAD. Given a hypernode, we identify *intra-class dimensions* as the union of the attributes of the entity node and the attributes of the literal edge of a given attribute node. *Intra-class measures* are then explored within each hypernode. The label of the attribute node is the name of the measure and the actual values of these measures are embedded in the attributes of the literal nodes. We distinguish here two types of graph aggregations: (1) Intra-hypernode aggregation, where literal nodes and edges of the same attribute node are merged, thus the dimensions is an attribute of the literal edges (e.g., the revenue of a given movie by language), (2) Inter-hypernode aggregation, where entity nodes could be merged (e.g., the revenue of all movies per city, period and language).

Example 4.3 (Analysis of the Revenue of a Movie)
Given the example of Figure 1.14, suppose an analyst needs to analyze the revenue of movies. Revenue is therefore considered as the name of the measure, for which the aggregation function is SUM. The values of the measures are stored within the literal nodes linked to the *Revenue* attribute node. The dimensions for the revenue measure are $Movie$, $Location$, $Period$,, and $Language$. Given these dimensions, we can aggregate the graph to examine the value of revenue by navigating through the dimension hierarchy of the $Location$ dimension from $City$ to $Country$ as shown in Figure 1.14-(a), or by rolling up to the level ALL of the language dimension as in Figure 1.14-(b). This is achieved by merging the corresponding literal nodes storing the measure values.

Another interesting feature of GRAD is the support of hierarchies, which

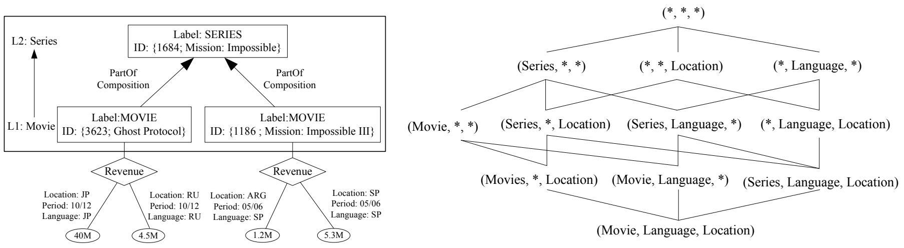


Fig. 1.15: Dimension hierarchy between classes

applies to both inter-class and intra-class dimensions as follows:

- Dimension hierarchy for intra-class dimensions: Within each dimension (e.g., attribute location of revenue), we might have an inner hierarchy (e.g., City, Region, and Country). Therefore, we can extend the lattice with these new possible aggregations as shown in Figure 1.14-(a).

- Dimension hierarchy for inter-class dimensions: Explored between distinct classes of nodes. Within GRAD, specific types of edges such as composition and aggregation could be explicitly defined. Therefore, classes of nodes related by these specific relationships belong to the same dimension with the hierarchy following the child-parent direction of these relationships. Figure 1.15-(a) shows the hierarchy of the movie dimension that is now composed of *Movie* and *Series* levels. The updated lattice is shown in Figure 1.15-(b).

4.3 End-to-End Computation and Analysis of Graph-specific Cubes

Graph warehousing is emerging as the field that extends current information systems with large graph management and analytics capabilities. Many approaches were proposed to address the graph data warehousing challenge [48, 103]. These efforts laid the foundation for multidimensional modeling and analysis of graphs. In this work, we extend the state-of-the-art on graph warehousing by introducing new types of graph cubes that leverage both the content and the topology of the graphs and expose topological and graph-structured insights. We present **TopoGraph**, a graph data warehousing framework that extends current graph warehousing models with new types of cubes and queries combining graph-oriented and OLAP querying. TopoGraph goes beyond traditional OLAP cubes, which process value-based grouping of tables, by considering in addition the topological properties of the graph elements. And it goes beyond current graph warehousing models by proposing new types of graph cubes. These cubes embed a rich repertoire of measures that could be represented with numerical values, with entire graphs, or as a combination of them. Given the proposed cubes, TopoGraph aims at providing answers to complex questions, asked in a business context, that require the analysis of both the content and the topology of the graph. Relevantly, TopoGraph is our proposal to overcome the current shortcomings of graph warehousing approaches resulting from our experience in real-life enterprise settings.

Multidimensional Structures on Graphs In TopoGraph, we distinguish three types of graph cubes, depicted in Figure 1.16:

- **Property Graph Cube:** captures only the content-based properties of graph elements. Dimensions and measures are a subset of the attributes of graph elements. It represents our baseline and is similar to a relational cube in that the measures do not capture the graph topology.
- **Topological Graph Cube:** captures the topological properties of graphs and represents them with numerical values. They are obtained by restructuring the topological multidimensional graph in all possible aggregations through the topological dimensions and/or by embedding and aggregating topological measures. Topological measures and dimensions are a subset of topological attributes such as community and PageRank.
- **Graph-structured Cube:** captures and represents the dimensional concepts using graphs. The graph-structured dimensions are dimensions whose values are represented as graphs. They express complex dimension values that could not be represented by numerical values and provide therefore a powerful selection means to examine non-trivial grouping of nodes or edges. The graph-structured measures are measures where the values are represented as graphs, which enables capturing and exposing insights and metrics structured as graphs.

Figure 1.17 depicts the coupled processes of (1) aggregation of graph cubes, and (2) generation of corresponding OLAP cubes and the mapping kept between them. This mapping is important, as the graph topology corresponding to each OLAP cuboid needs to be preserved to compute the topological measures such as PageRank. The measures could afterward be loaded into the OLAP cubes for further multidimensional analysis.

Example 4.4 (Deriving Popularity OLAP cube from the Graph cube)
Given the Twitter multidimensional graph, we design a lattice, as shown in Figure 1.17. Each point in the lattice corresponds to a

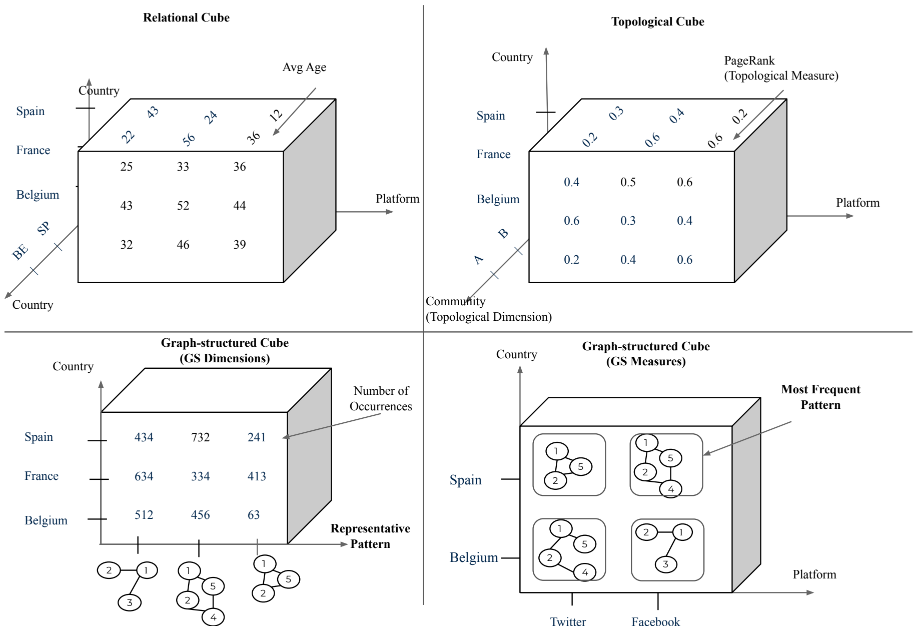


Fig. 1.16: Types of Derived OLAP Cubes

graph cuboid. For simplicity, we consider the dimension attributes: $\{Location, Community, Platform\}$, while ignoring the hierarchies of the location dimension. We highlight two particular aggregations: (1) node-only aggregations (i.e., only dimensional attributes from user nodes are kept not fully aggregated as in ($\langle Location, Community, *\rangle$, $\langle Location, *, *, *\rangle$, and $\langle *, Community, *\rangle$), and (2) edge-only aggregation as in ($\langle *, *, Platform\rangle$). The fact analyzed is the popularity of users. The measure is PageRank, computed by applying the PageRank algorithm in the social network following the edges labeled *Connected*.

The topological and graph-structured cubes enable structuring the multidimensional space in a novel way capturing graph elements that are connected in a complex manner. Another main benefit of graph cubes is that they minimize the information loss, as they keep the graph structure after being computed or aggregated. A common assumption is that each graph cuboid can be loaded into a relational OLAP cube. This is true for content-based graph cubes. However, loading a graph cuboid into a relational cube causes the loss of the graph structure. Therefore, current warehousing systems are not designed to support this type of cubes, which further motivates the need for developing native graph warehousing systems.

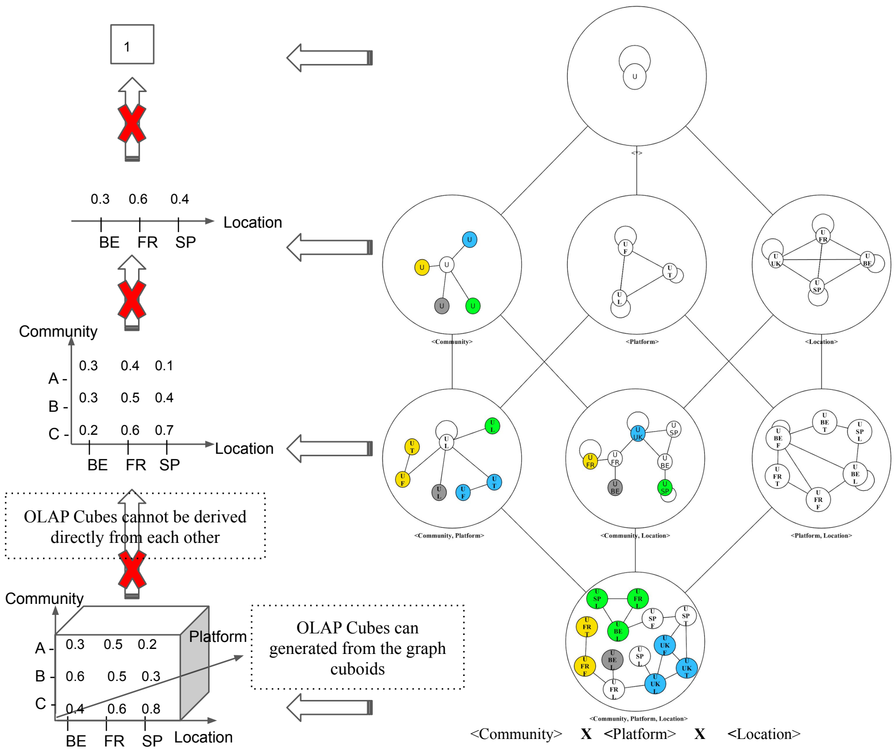


Fig. 1.17: OLAP Cube Generation form Graphs

Integrity constraints Multidimensional aggregation of a property graph is the operation of consolidating a set of graph elements into a single one located at a higher level of the lattice. Two constraints need to be enforced when building graph cubes: (1) correct aggregation of the graph cuboids, and (2) correct placement of the graph measures within the multidimensional space. To ensure a correct aggregation of cube measures along dimension hierarchies, the graph aggregation should satisfy three constraints (1) completeness, so that every graph element is associated to at least one dimension level, (2) disjointness, so that each graph element is included at most once to create an aggregate entity and could not belong to more than one dimension level at once, and (3) compatibility between the aggregation algorithm and the aggregate graph elements to prevent non-meaningful operations. These constraints were defined on the lines of the summarizability constraints introduced by Lenz and Shoshani in [80], but with the notable difference of capturing the graph topology (such as aggregation using graph patterns) and ensuring the compatibility with graph measures (such as preventing the sum of PageRank of aggregated users). Multidimensional integrity constraints on graphs were completely overlooked by current approaches. To the best of our knowledge, our framework is the first to define and guarantee the multidimensional integrity constraints on graphs.

OLAP Analysis of Graph Cubes OLAP analytics supports interactive and complex queries over large volumes of data, from different perspectives and through different hierarchical levels. Thus, enabling analysts to highlight the data item of interest, and then drill down to the underlying data from which it has been created. This could help in decision support scenarios such as the measurement or comparison of the business performance across the different dimensions. In this section, we describe a set of algebraic operators for OLAP querying of multidimensional graphs. We consider the graph cubes defined and computed in the previous sections as the fundamental construct of the multidimensional model. The graph cubes are the operand and the return type of all OLAP operations. We illustrate the application of each operation on a graph cuboid and its corresponding OLAP cube. In addition to the cuboid and crossboid operations that were defined in the literature [143], we present the major OLAP operations applied on graph and OLAP cubes.

- *Multidimensional selection:* restricts the graph to a subgraph where all nodes and edges match the selection pattern.
- *Roll-up/Drill-down:* aggregates the graph along its dimensions. This operation modifies the granularity of the graph using a many-to-one relationship which relates instances of two levels in the same dimension hierarchy, corresponding to a part-whole relationship. Roll-up performs

structural changes to the graph and generates a new graph placed at the next level of the dimension hierarchy while respecting the summarizability integrity constraints. Roll-up is implemented in three phases (1) first a selection of graph elements matching the aggregation pattern that describes the graph elements, then, (2) the graph aggregation to shape the graph at the aggregation level, and finally (3) measures are (re)computed and placed on the aggregate graph.

- *Drill-across/Projection:* This operation changes the subject of analysis of the cube using a one-to-one relationship. The n-dimensional space remains the same, only the cells placed on it change. With this operation, different measures are placed on the same multidimensional space. Projection is the reverse operation of a drill-across. It selects a subset of measures of interest to be studied within the multidimensional space.

Example 4.5 (Combining Topological and Graph Structured Measures) Figure 1.18 shows an example of a drill-across between a topological and a graph-structured cube. Both cubes are placed in a cube having as dimensions $(\langle Community \rangle, \langle Country \rangle)$. The first is a graph-structured cube containing representative communities, and the second is a topological cube containing PageRank. Using drill-across, the measures from the two cubes could be embedded in the same cells and analyzed within the same cube. Inversely, a projection would for instance remove the measure representative community from the cube to focus only on studying the PageRank.

Implementation We implemented and experimentally validated TopoGraph's efficiency with different public real-world datasets such as Twitter, DBLP, and LiveJournal, and in different sizes up to 34 million edges. We compared the cuboid generation and aggregation time for each dataset at different levels. An illustration of the use of TopoGraph for the analysis of a social network is described as depicted in Figure 1.19.

4.4 Architecture of Graph Data Warehouse

The topological properties of graphs are of big potential to decision-making systems. They supply these systems with a new class of complex structural business facts and measures that could be explored for making more accurate decisions in data-driven organizations. In current information systems, Business Intelligence (BI) systems are critical for strategic decision-making. Graph BI, in particular, is emerging as the BI field that extends current BI systems with graph analytics capabilities. It enables graph-based insights such

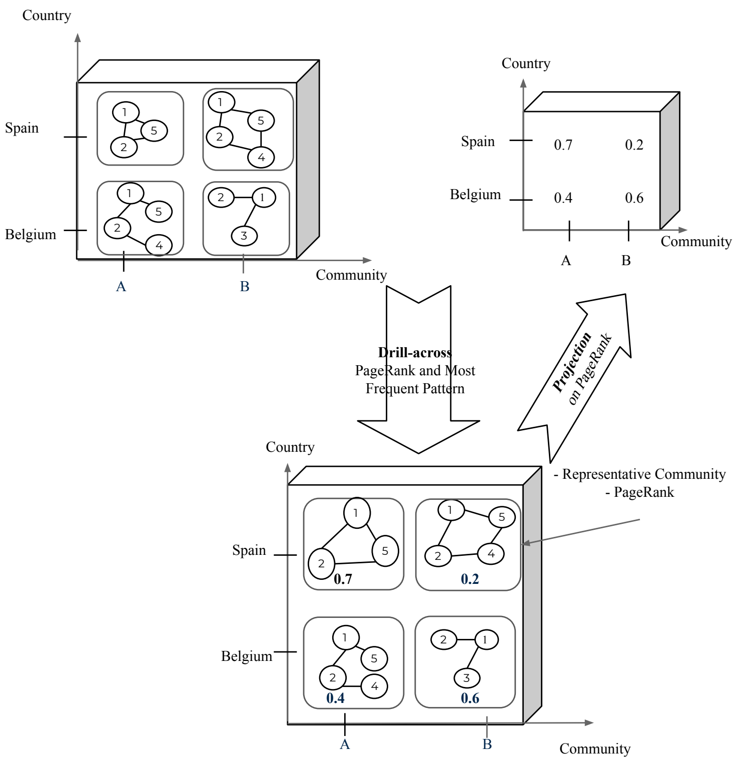


Fig. 1.18: Drill-across and Projection

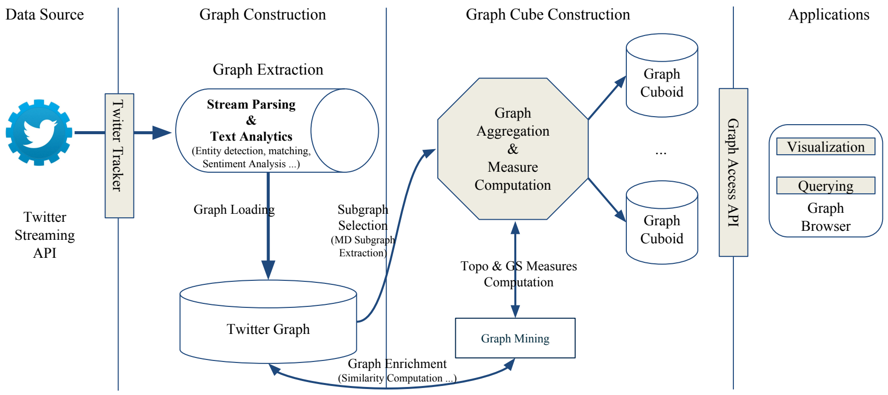


Fig. 1.19: Social Networks Warehousing

as detection of popular users or communities in social networks, or revealing hidden interaction patterns in financial networks. The structure-driven management and analytics of graph data call for the development of novel data models, query processing paradigms, and storage techniques. Therefore, as motivated by multiple research lines [43, 83], current BI and analytics systems need to be extended to efficiently support warehousing [38], processing [120], mining [119] and OLAP analysis [103] of the graph structural and content-based information.

In this work, we surveyed the state-of-the-art on graph BI and analytics and proposed an architecture of a Graph BI and Analytics platform augmented with machine learning capabilities, which lays the foundations for promising future research directions. We note that, while adopting a similar template as the traditional BI systems (i.e., it preserves the familiar data analytics workflow), graph BI extends current systems with graph-aware components that deliver graph-derived insights. Hence, we discussed the main topics related to graph warehousing, surveyed the existing frameworks for graph analytics, and identified future research directions, and positioned them within a unified architecture of a graph BI & analytics framework. Figure 1.20 provides an overview of the different components of the envisioned graph warehousing system. We highlight our contribution to the graph warehousing topic by positioning part of the publications made on this thesis in this proposed architecture.

We describe the main building blocks of a graph BI framework as follows:

- Graph Extraction (1): graph data is extracted from different data sources that could be in various formats and flowing at dynamic rates such as graph streams. The data is cleaned to capture entities that satisfy the quality constraints.

- Graph Construction & Enrichment (2): The captured graph data is integrated and formatted according to a given graph model. Natural Language Processing (NLP) algorithms could help in the automated extraction and construction of multidimensional graphs from unstructured data such as text.

- Graph Data Warehouse (3): The cleansed and integrated data is natively stored and managed as a multidimensional graph in the graph warehouse. The graph warehouse provides a suitable backbone for natively analyzing graphs with BI tools such as graph OLAP and graph mining.

- Cube Design and Computation (4): Given a multidimensional graph, the graph cubes enable the computation and the aggregation of corresponding graph cuboids. Once the required graph cuboids are computed, the result is persisted in the corresponding data marts. To leverage graph properties, graph cubes embed graph-structured measures

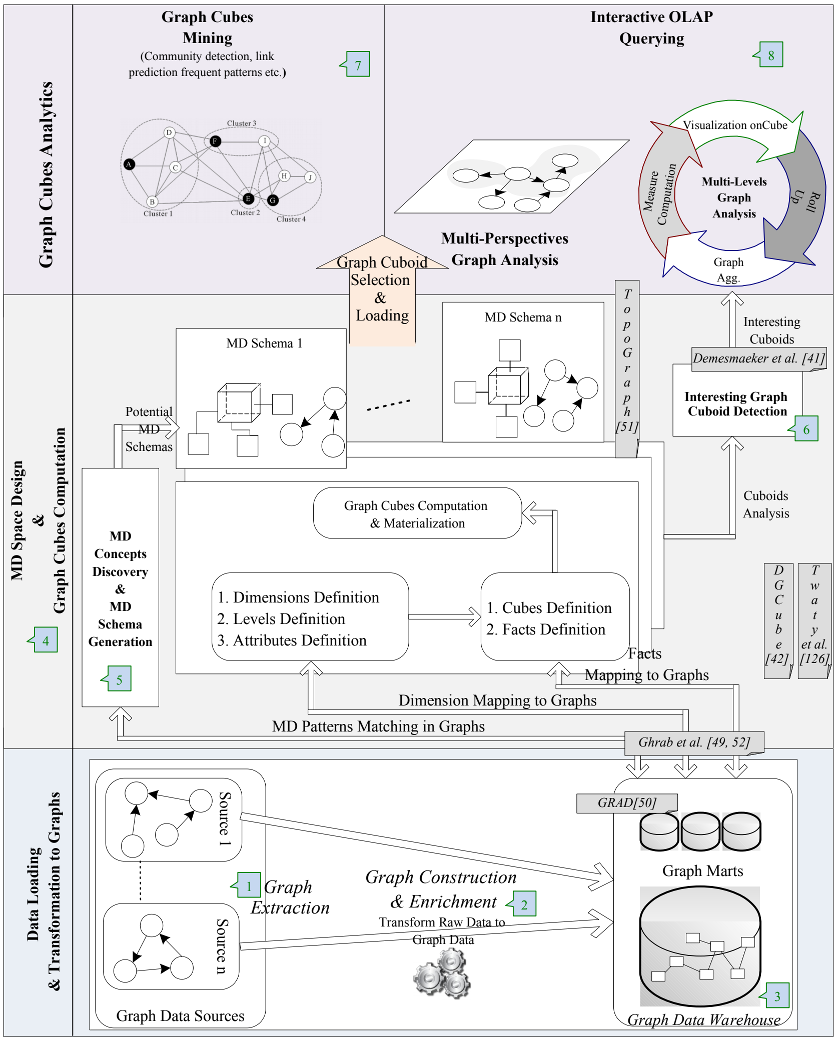


Fig. 1.20: Architecture of a Graph BI & analytics framework

and dimensions. There is a need for cube computation and aggregation libraries capable of efficiently handling graphs.

- Discovery of multidimensional concepts and definition of potential multidimensional schemas (5): Multiple multidimensional schemas could be built from the same graph warehouse to satisfy the various analysis needs. Interesting graph entities might be hidden in the large data sources. Therefore, there is a need for novel graph-aware approaches that enable automatic detection and extraction of multidimensional concepts from large complex graphs. This will help end-users cope with the complexity and large volume of graphs, and expose potential interesting discovery to decision-makers.

- Assistance with the analysis and synthesis of graphs (6): Given the complexity and large size of the initial graph, there is a need for intelligent modules capable of performing an automated preliminary analysis of the graph to guide the analyst during the exploration of the graph cubes. The goal is to enable self-service BI and facilitate complex tasks such as the extraction of meaningful graph summaries or the discovery of interesting phenomena in graph cuboids.

- Mining (7) and querying OLAP cubes (8): Complex and interactive OLAP analysis and mining of graph cubes are performed at this phase. There is a need to develop graph OLAP engines that support graph-structured cubes. Besides, Online Analytical Mining of graph data is a promising research direction to empower graph OLAP with mining capabilities. Graphs are dynamic and enabling OLAP on evolving networks by analyzing changing facts and dimensions will help in understanding the structural and informational evolution of networks.

5 Summary

Table 1.1 summarizes the objectives and contributions of this thesis and refers the reader to the related chapters and papers. The following chapters are self-contained, but each builds on the previous, so we advise to read them in the order.

Research Question	Contributions	Related Chapters and Papers
How to model a graph to make it ready for warehousing and OLAP analytics?	We propose GRAD, a native graph database model designed primarily for graph data warehousing. GRAD aims to provide analysts with a set of simple, well-defined, and adaptable conceptual components to support the main warehousing tasks. It brings the power of established modeling techniques to graph data modeling to better accommodate graph warehousing. GRAD introduces a set of data structures specifically annotated to facilitate the identification of multidimensional concepts such as dimension hierarchies. Besides, a set of integrity constraints are proposed to ensure consistency. Finally, the model is equipped with content and topology-aware algebraic operators inspired by a relationally complete graph algebra and designed to support OLAP operations.	Chapter 3 [50, 52]
How to design a multidimensional model aware of the topological characteristic of the graph, and how to leverage these properties in OLAP analysis?	• We define the multidimensional concepts for graph data, and propose novel techniques for building graph cubes from property graphs. We examine the additional graph cubes brought by GRAD and illustrate their support for advanced concepts such as dimension hierarchies. • We formally define novel types of graph cubes that capture both content and topological properties of heterogeneous graphs. • We discuss the big gap between graph cubes and relational OLAP cubes and motivate the need for dedicated graph warehousing systems. • We define the algebraic OLAP operations for the new graph cubes, and illustrate their application for querying the topological information embedded in the graphs. • We propose a prototypical implementation of TopoGraph and describe the architecture and the API of a social network analysis framework built with it.	Chapter 4 [49] and Chapter 5 [51]
What are the main building blocks of a graph warehouse, and how could they be unified in a comprehensive architecture?	We design the architecture of graph warehousing and analytics platform, similar in its template to traditional systems, but aware of graph specificity and augmented with machine learning capabilities. We overview the main components of the proposed graph warehousing system and highlight our contributions within.	Chapter 6 [48]

Table 1.1: Summary of the Objectives and Contributions

Chapter 2

Related Work

In this chapter, we present the current literature in the field of graph management and warehousing. The goal is to identify the open challenges and gaps in the design and optimization of graph data warehouses.

1 Graph Database Management

With the new Big Data requirements and NoSQL movement, interest in graph management and analytics increased, and motivated researchers to address graph-specific issues not efficiently solved by traditional database systems. In particular, graph database management gained a lot of momentum in the data management community in recent years [10, 20, 22]. Two approaches are widely used for modeling and managing graph data. The first is leveraging the current models, mainly the relational model. The second is to build native graph data models and database engines. We discuss the state-of-the-art on each of these directions.

Graphs on Relational Systems The advantage of this approach is that once data is loaded in a relational system, it gains the benefits of the well-established relational model, and smoothly integrates with the wide range of relational platforms. The relational model was designed to handle structured data such as records and transactions. A graph could be modeled as a collection of tables of nodes and edges. Each node is uniquely identified by its primary key, and edges are attached to their source and destination nodes using their primary keys as foreign keys. Many studies evaluated the readiness of alternative models, and essentially relational systems for supporting graph workloads [32, 61, 133]. For example, Vicknair et al. evaluated the performance of Neo4j and MySQL for traversal and count queries [133].

Pobiedina et al. benchmarked pattern matching queries in Neo4j and PostgreSQL among other databases [99]. The reported performance results are often mixed, as relational engines might perform better for large scans and aggregations, while graph engines are more optimized for traversal oriented analytics. From the modeling perspective, graphs are flexible and do not usually follow a fixed schema, if they were to be stored in a relational engine, the schema will have to change a lot. Supporting changes, extensions, or integration of different schemas are difficult, costly, and manual operations. Due to the fundamental difference between the two models, the transformation of graph data to a relational model is a manual, complicated process. It incurs a high risk of information loss during the transformation process. The SQL query language and processing engines are optimized to perform table scans instead of traversals. They are not suited to capture the topology of the graph with queries such as finding the shortest path or neighborhood discovery. Graph operations such as finding cycles, or matching complex graph patterns introduce a heavy workload, especially for highly connected tables [96].

We conclude that the relational model and its implementations fall short of meeting the requirements for (1) intuitive data modeling, (2) topology-aware graph querying (such as path retrieval and comparison, and graph pattern matching), and (3) traversal-optimized performances. This motivates the need for native graph data management systems.

Native Graph Databases In recent years, the trend in graph data management has shifted to the development of native, relationship-oriented graph databases. Native graph databases introduce custom optimizations for the storage and querying of graph data by using specific graph structures and supporting the querying with a set of special operators, algorithms, and indexing techniques. A comparison of modern graph database models is provided on [6, 20].

Most native graph databases implement the property graph model or a variation of it. This model was first described by Rodriguez and Neubauer in [109] to denote directed, labeled, and attributed multi-graph. It was later formally defined by some works [33, 110], but most notably in [7]. Each real-world entity is represented by a node that contains its label and properties. The label denotes the type of the node (i.e., the class to which it belongs). Relationships between entities are represented using edges. The flexibility of property graph models allows the representation of rich structural properties, such as hierarchies and dependencies. Property graphs were introduced in the database community to store schemaless data (due to their flexibility to absorb any semantics and attach data with metadata), where relationships are first-class citizens, and the data is represented as perceived, without the need

to map it to intermediate representations. As shown by multiple studies, this makes the data model more straightforward to design, and the queries are more intuitive to formulate [61, 62, 63].
From a performance perspective, graph databases are optimized for graph traversals. The cost of traversing an edge is constant, and the overall cost of arbitrary navigation through the graph is much lower than the equivalent joins in relational tables [61, 115]. Subsequent implementation aspects such as physical graph storage, indexing, and retrieval which are specifically tuned for graph workloads lead to better performances, especially for queries requiring multiple joins, or containing complex patterns such as cycles [59, 125, 144]. However, they perform worse than the relational-based engines for analytical queries that perform scans over the whole graph.
In industry, a plethora of graph database tools are developed, with various modeling and querying options. Most of the current databases are oriented for OLTP processing of graph data. They support similar features to relational databases such as ACID properties, indexing, and query languages. In the software market, established vendors are aware of the potential of native graph solutions and have already developed many graph databases such as Oracle Spatial and Graph [64, 105], Microsoft GraphEngine, SAP HANA [112], IBM Graph, and Amazon Neptune.

Graph Query Languages A key factor in the success of relational databases is the relational algebra and its mapping to the SQL query language. Multiple native graph query languages were developed to efficiently answer graph-oriented queries [9, 22, 136]. However, no standard graph query language is yet available. Tian et al. [124] proposed a technique K-SNAP for drilling up and down across different aggregation levels on heterogeneous graphs. Zhao et al. studied the issue of querying large heterogeneous information graphs. Their goal is to define an SQL-like declarative language specific to information networks. They designed two operators, P-Rank and SPath, for approximate and exact subgraph matching respectively [142]. GraphQL[60] proposed a relationally complete algebra, and as reported by Lee et al. [78], was the only algorithm to complete the typical graph queries tested on their comparison (subgraph, clique, and path queries), although at a slower performance compared to some of the other tested algorithms. An interesting characteristic of GraphQL is that it provides a mapping between the algebraic expressions and FLWR expressions from XQuery. The query language is declarative, graph-oriented, and suited for semi-structured data. Multiple query languages were proposed to enable the querying of property graphs [9], such as Cypher [46], GCore [8], PGQL[129] and Gremlin [108]. Cypher is an SQL-like declarative language, that uses isomorphism-based no-repeated-edges bag semantics. It was introduced by Neo4j and is centered around pat-

tern matching enriched by built-in algorithmic libraries. Gremlin is a graph traversal language, built using Groovy, introduced by Apache TinkerPop3, that uses the homomorphism-based bag semantics. Given the diversity of the property graph query languages, a recent initiative is led for defining GQL [1], an ISO project to design a standard query language for property graphs by combining the best of the main property graph query languages (Cypher, G-CORE, and PGQL).
Integrity constraints for graphs, and particularly property graphs, is a less explored topic [7, 100, 101], and focus mainly on schema-instance consistency, although other constraints such as attribute uniqueness or cardinality constraints were discussed.

Comparison Most of the graph database models presented in the literature were designed to handle OLTP and not OLAP workloads. They introduced a specific set of data structures, algebraic operators, and constraints. However, they were designed for operational and not analytical purposes, and are not well-equipped for multidimensional analysis of graph data. For example, the commonly used property graphs are loosely constrained, which makes it difficult to automate tasks such as the identification of potential multidimensional concepts and extraction of graph cubes. In GRAD, we suggested a natural representation of graph data, with a focus on structural properties' analysis. GRAD provides a complete model that facilitates multidimensional modeling by proposing (1) analytical graph structures that capture rich semantic such as encapsulation and hierarchies, (2) a set of operators to manipulate the graph and compute or extract graph specific metrics, and (3) a set of rules to enforce and preserve the integrity of the graph data through the different integration and transformation operations. Thus, GRAD could be seen as a specification of the core elements a graph database model needs to support graph warehousing and OLAP workloads. Table 2.1 compares GRAD to the current graph models from a database modeling perspective. It presents an overall view of how GRAD contextualizes with regards to main graph models such as property graphs, and new graph models such as RDF*. Note that GRAD was designed in 2013 to identify the required foundations a model needs to tackle multidimensional analysis in a full-fledged manner. Notably, aspects such as integrity constraints are not directly covered in current graph database models (even if novel solutions such as SHACL could lay foundations to define the multidimensional constraints required). In conclusion, property graphs and RDF* combined with SHACL, are the current best options to implement GRAD in real scenarios. Yet, there is still a need for more flexible options to augment data warehousing with graphs and to build mature systems like in the relational setting, which we discuss in the next section.

Criteria	**GraphQL (2008)**	**Property Graphs (2010)**	**RDF (2004)**	**RDF* (2014-2017)**	**GRAD (2013)**
Graph Model	Collection of labeled attributed graph	Directed, labeled attributed multi-graph	RDF triples	Extended RDF triples with nodes and edge attributes	Directed, labeled, attributed multi-graph with multidimensional semantics
Data Structures	nodes, edges	nodes, edges	subject, predicate, object	subject, predicate, object, attributes	nodes, edges, hypernodes
Algebraic Operators	- Selection - Cartesian product - Join - Composition - Set operators	- Navigational Pattern matching - Negation - Union	SPARQL Algebra	SPARQL* Algebra	- Selection - Cartesian product - Composition - Set operators - Structural join
Query Language	GraphQL	Cypher	SPARQL	SPARQL*	N/A
Integrity Constraints	N/A	- Schema-instance - Mandatory and optional properties/edges - Unique attributes - Cardinality constraints	SHACL	SHACL	- Entity integrity - Semantic constraints
Exemplary DB Systems	Native Graph DB	Native Graph DB	RDF Store	RDF* Store	N/A
	Sones GraphDB	Neo4j	Stardog	GraphDB	

Table 2.1: Comparison of Graph DB Models

2 Graph Warehousing

A lot of research has been devoted to extend data warehousing and OLAP technology beyond the relational systems [38, 39]. Various efforts were led to support other data formats such as text [84], multimedia [68], and graphs [103]. Multiple architectures and systems were proposed in the literature to integrate graph data into business intelligence systems. BIIIG [98] is a framework for business intelligence on graphs that focuses on the use of the graph's flexibility in data integration. It enables integrating and referencing heterogeneous data from different sources. Li et al. [82], proposed conceptual models for designing and querying graph data warehouse systems. In [48, 120], authors suggested a novel architecture for graph BI systems, that leverages large graph mining and warehousing. Our work in this thesis goes in-line with these research directions, and attempts to provide a foundation for extending decision-making systems, and particularly OLAP, with graph analytics capabilities, while paying particular attention to the few cases of possible correspondence between graph and ROLAP cubes.

Graph OLAP Early research in graph warehousing started with the Graph OLAP model, which set the first foundations for multidimensional modeling and analysis of graphs. Graph OLAP supports the multidimensional modeling and analysis of a collection of homogeneous graph snapshots [28]. Two types of modeling and analysis are performed: (1) informational and (2) topological. In informational OLAP (I-OLAP), the dimensions are attributes of the graph snapshot. The aggregation of the graph is performed by overlaying and merging a set of graph snapshots that share the same dimension values. The analysis consists in edge-centric snapshot overlaying. Thus, only the edges are merged and changed, with no changes made to the nodes. In topological OLAP (T-OLAP), the attributes of the nodes are called topological dimensions. The aggregation consists of merging nodes and edges by navigating through the nodes' hierarchy. T-OLAP was discussed in a more detailed framework for topological OLAP analysis of graphs [102]. Their paper discussed the topological aggregation of the graph following the OLAP paradigm. They presented techniques based on the properties of the graph measures (T-Distributiveness and T-Monotonicity) for optimizing measures computations through the different aggregation levels. Another multidimensional model [19], similar to Graph OLAP, was proposed and considered the dimensions as the labels of the edges, and presented a set of analytical graph-based measures relevant for OLAP analysis of graph data. HMGraph introduced a data warehousing model for heterogeneous graphs focusing on edge-based dimensions [140]. It enriched the informational and topological dimensions with the entity dimension and the rotate and stretch operations along with the notion of metapath to extract subgraphs based on edges traversals.

Graph Cube The second family of frameworks focused on the efficient computation and extending the querying of OLAP cubes derived from multidimensional graphs. [143] introduced the first framework that coined the term GraphCube. The authors defined a multidimensional graph from a single, homogeneous attributed graph, by choosing a subset of the attributes of the nodes to be the dimensions. The aggregate graph itself is the measure. The graph cube is obtained by restructuring the initial graph in all possible aggregation. The framework introduced two types of queries: (1) the cuboid query, which generates 2^n aggregate graphs, and (2) the crossboid query, which analyzes the interrelationships between different graph cuboids. Many frameworks were proposed afterward to (1) support more general graph models, (2) new types of multidimensional structures, (3) novel OLAP queries, and (4) custom materialization strategies. Pagrol introduced a parallel graph cube framework that extended the original GraphCube model by defining the Hyper Graph Cube model that considers the attributes of the nodes and edges as dimensions [135]. Both GraphCube and Pagrol designed various materialization policies to speed up the computation and analysis of graph cubes. However, both GraphCube and Pagrol were still limited to homogeneous graphs. The graph model was later extended with a framework for building OLAP cubes supporting heterogeneous attributed graphs and dimension hierarchies [49]. The TSMH framework introduced the concept of relation path to guide the graph aggregation and building two new types of cubes: Entity Hyper Cube and Dimension Cube [134]. P&D Graph Cube extended the graph cube model by introducing the concept of path and dimension aggregate networks, along with their materialization strategies [137]. A multidimensional model for directed multi-hypergraphs and its query language were proposed in the literature, along with an implementation using Neo4j [53]. Other research lines focused on applying graph warehousing for specific domains such as the analysis of bibliographic data [86], or business process data [17, 128]. For example, distributed OLAP analytics of process execution data represented as graphs was tackled by designing a Hadoop-based framework [17]. Thus enabling, multi-level and multi-perspective analysis of large volumes of business process data represented as graphs.

Comparison Existing work for OLAP analysis on graphs provided a foundation for OLAP cubes computation and querying on graphs. Table 2.2 summarizes the related work from modeling and implementation perspectives. As shown in the table, in most state-of-the-art frameworks, the only measure that is examined is the aggregate graph itself, and the dimensions are a set of attributes or paths. TopoGraph extended these frameworks by supporting the general case of the property graph model and proposing new types of graph cubes that embed novel types of measures and dimensions. For these new graph cubes, the algebraic OLAP operators required for their analysis were defined and illustrated. Further, to the best of our knowledge, TopoGraph is the first framework to discuss the multidimensional integrity constraints on graphs. These constraints are a key concept that guarantees the correctness and soundness of graph cube construction aggregation. Moreover, for each of the introduced cubes, the possible correspondence with ROLAP cubes was discussed to bridge the gap between the two communities and favor the integration of graphs within relational OLAP frameworks. However, as discussed, combining graphs and traditional cubes is, in the general case, difficult. For this reason, we highlighted the need to keep researching specific graph warehousing tools that preserve the graph structure and benefit from graph-specific modeling and processing. Finally, TopoGraph still requires further physical optimizations, such as custom graph indexing and materialization, that capture the specific nature of topological and graph-structured cubes [59, 144].

References	Graph Model	Multidimensional Model				Querying	Scalability
		Dimensions	Measures	Cube	IC		
Graph OLAP [28]	Collection of Homogeneous Labeled Graph Snapshots	Info-Dims & Topo-Dims	Traditional, Topological, Aggregate Graph	Graph Cube	-	Roll-up/Drill-down/Slice-dice	Centralized
GraphCube [143]	Homogeneous, Node Attributed	Node Attributes	Aggregate Graph	Graph Cube	-	Cuboid & Crossboid	Centralized
HMGraph [140]	Heterogeneous, Attributed	Information, Topological, Entity	Aggregate Graph	HMGraph Cube	-	Rotate, Stretch, Roll-up, Drill-down	Centralized
Distributed Graph Cube [42]	Homogeneous, Node Attributed	Node Attributes	Aggregate Graph	Graph Cube	-	Cuboid & Crossboid	Distributed (Spark)
Pagrol [135]	Homogeneous, Attributed	Node and Edge Attributes	Aggregate Graph	Hyper Graph Cube	-	Roll-up & Drill-down	Distributed (Hadoop)
TSMH [134]	Heterogeneous, Node Attributed	Node Attribute & Meta-Path	Aggregate Graph	Entity Hyper Cube, Dimension Cube	-	Roll-up, Drill-down, Attribute Transformation	Distributed (Spark)
Gómez et al. [53]	Labelled Directed Multi-hypergraphs	Dimension Graphs	Aggregate Graph	Graphoid	-	Climbing and Aggregation, Roll-up, Drill-down, Slice, Dice	Centralized
P&D GraphCube [137]	Heterogeneous, Attributed	Node Attribute, Relation Path Set	Aggregate Graph	P&D Graph Cube	-	Entity, Edge, Topological Structure	Distributed (Hadoop, Spark)
[74]	Homogeneous, Attributed	Node and Edge Attributes	Aggregate Graph	Graph Cube	-	Cuboid	Distributed (Spark)
TopoGraph	Property Graph	Content, Topological, Graph-structured	Content, Topological, Graph-structured	Content, Topological, Graph-structured	X	Selection, Roll-up, Drill-down, Drill-across	Centralized

Table 2.2: Comparison of Graph Cube Frameworks

Chapter 3

GRAD: A Database Model for Advanced Graph Analytics

The paper has been published as a preprint in ArXiv. The layout and content of the paper have been revised.
Reprinted, with permission, from Amine Ghrab, Oscar Romero, Sabri Skhiri, Alejandro Vaisman, and Esteban Zimányi, GRAD: On Graph Database Modeling, arXiv preprint:1602.00503 (2016).

Abstract

Graph databases play a key role in the emerging graph warehousing ecosystem. They are a fundamental technology underpinning a multitude of domains and applications where traditional databases are not well-equipped to handle complex graph data management and analysis. However, the current graph database models and systems are mainly designed for OLTP workloads and do not provide the required foundations for graph warehousing.
In this chapter, we introduce GRAD, a native graph database model tailored for graph data warehousing. GRAD presents new data structures to support graph historization and integration, a set of well-defined constraints to guarantee the graph integrity, and graph algebra to facilitate OLAP analysis. Thus, the main contribution of GRAD is an exhaustive list of the required components a graph model must fulfill to support graph data warehousing. Relevantly, GRAD was defined following Codd's data model definition.

1 Introduction

Graph warehousing refers to the process of collection, integration, historical storage, and multidimensional analysis of graph data. In this chapter, we address the need for a native graph database model tailored for graph warehousing. Indeed, databases play a central role in data warehousing to ensure the efficient storage and retrieval of data. At their core, databases rely on a database model that provides the necessary theoretical foundations. A database model, as defined by Codd [34] consists of a set of (1) data structures, (2) integrity constraints, and (3) manipulation operators. In this work, we chose to design a new native graph database model that implements Codd's definition and suits warehousing. Native graph models have various advantages over non-native model such as (1) enabling natural domain modeling since the graph structures are similar to the way end-users perceive the domain, (2) querying is user-friendly and less error-prone since the operators target directly the network structure, and can examine specific graph measures, (3) advanced concepts are easier to implement, for example, assertions are not implemented on current relational databases and can be defined and implemented in graph databases by using graph patterns.
Thereby, we introduce GRAD (GRAph Database model), a native graph database model designed primarily for graph data warehousing. GRAD aims to provide analysts with a set of simple, well-defined, and adaptable conceptual components to support the main warehousing tasks. Indeed, GRAD introduces a set of data structures specifically annotated to facilitate the identification of multidimensional concepts. For example, relationships could be annotated as *aggregations* which helps to identify dimension hierarchies. Moreover, design choices such as node attribute reification, and the unique node identifier constraint are made to facilitate graph historization, a fundamental task in graph warehousing. Thus, the integration of different versions of a GRAD graph translates to a graph union operation that does not require changes in node attributes. Besides, a set of integrity constraints are proposed to ensure consistency. Also, the model is equipped with content and topology-aware algebraic operators inspired by a relationally complete graph algebra and designed to support OLAP operations. Overall, GRAD aims to bring the power of established modeling techniques to graph data modeling to better accommodate graph warehousing.

In summary, our contributions in this chapter are:

- A native graph database model that introduces advanced data structures to capture the common notions present in established conceptual data modeling languages. GRAD structures natively support complex semantics such as encapsulation and composition to simplify data integration and multidimensional modeling tasks.

- A set of rules to enforce and preserve the integrity and consistency of the graph data, with a focus on graph entity integrity and semantic constraints.
- A collection of algebraic operators to enable online graph querying and analysis while preserving integrity constraints. The algebra is defined along the lines of relational algebra and inherits at least the same expressive power.

The remainder of this chapter is organized as follows. In the next section, we introduce the property graph model and the example used in the chapter. Then, we present the advanced data structures of GRAD where we project the common notions form conceptual modeling languages into graphs. We then define the different integrity constraints ensuring the graph database integrity. Following this, we present the GRAD algebra for querying and manipulating the graph data.

2 Property Graphs

Graph data are usually represented as collections of nodes and edges. Nodes represent entities and edges relate pairs of nodes, and are used to represent different types of relationships. Nodes and edges might be labeled, and have a set of properties represented as attributes. Many current graph databases represent graphs using the property graph model. Property graphs were first introduced by Rodriguez and Neubauer to describe a directed, labeled and attributed multi-graph [109]. Recently, more formal definitions were proposed in the literature [7, 22], which we adapt as follows:

Definition 3.1. [***Property Graph***] *Given a finite set of labels $\mathcal{L}$, a set of property keys $\mathcal{K}$ and a set of values $\mathcal{N}$, a property graph is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H})$ where:*

- *$\mathcal{V}$ is a finite set of nodes;*
- *$\mathcal{E}$ is a finite set of edges, such that $\mathcal{E} \cap \mathcal{V} = \emptyset$;*
- *$\rho : \mathcal{E} \to \mathcal{V} \times \mathcal{V}$ is the function that maps each edge to its pair of nodes, with $\rho(e) = (u, v)$ denoting a directed edge from u to v;*
- *$\mathcal{F} : \mathcal{V} \cup \mathcal{E} \to \mathcal{L}$ is the function that assigns to each node (resp. edge) a label from the set of labels $\mathcal{L}$;*
- *$\mathcal{H} : (\mathcal{V} \cup \mathcal{E}) \times \mathcal{K} \to \mathcal{N}$, is a partial function assigning to each node (resp. edge) the value of its attribute, such that, for a node (resp. edge) $v \in \mathcal{V}$, the function $\mathcal{H}(v, k_i)$ returns the value $x \in \mathcal{N}$ of its $i-th$ attribute identified by the key $k_i \in \mathcal{K}$.*

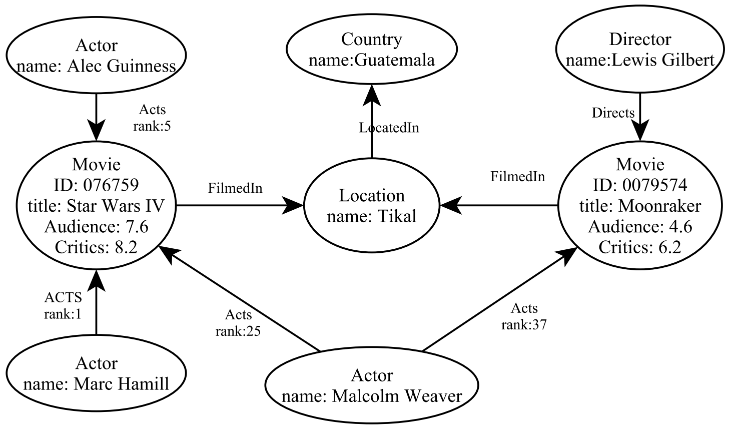


Fig. 3.1: MovieLens represented with Property Graphs

Many graph databases are built on top of property graphs (e.g., Neo4j and SAP HANA). However, property graphs do not define a full database model but rather describe basic graph structures oriented for OLTP instead of OLAP workloads. A native and comprehensive graph database model optimized for graph OLAP analysis and compliant to Codd's definition [34] is yet to be designed. We illustrate the notions introduced in this chapter using data from the *MovieLens + IMDb/Rotten Tomatoes* dataset, which was first published in the HetRec2011 workshop [25]. In the sequel, we refer to this dataset simply as MovieLens. The dataset integrates information about movies from MovieLens with their corresponding web pages at Internet Movie Database (IMDb) and Rotten Tomatoes movie review systems. We represent this dataset using property graphs as illustrated in Figure 3.1. The network contains information about movies classified by genres and filmed across multiple locations. Movies have different attributes, such as year of release, titles, ratings, and scores from different communities, and are linked to directors and actors (ranked by importance). Users provide ratings and tagging for the movies they watched.
Figure 3.2 illustrates how GRAD could be used to integrate those different movie graphs into one, similar to the integration phase of ETL. The notations and data structures introduced by GRAD are presented in the next section.

3 GRAD Structures

Multiple conceptual data models exist in the literature to support and standardize the representation of different domains and applications. For each of

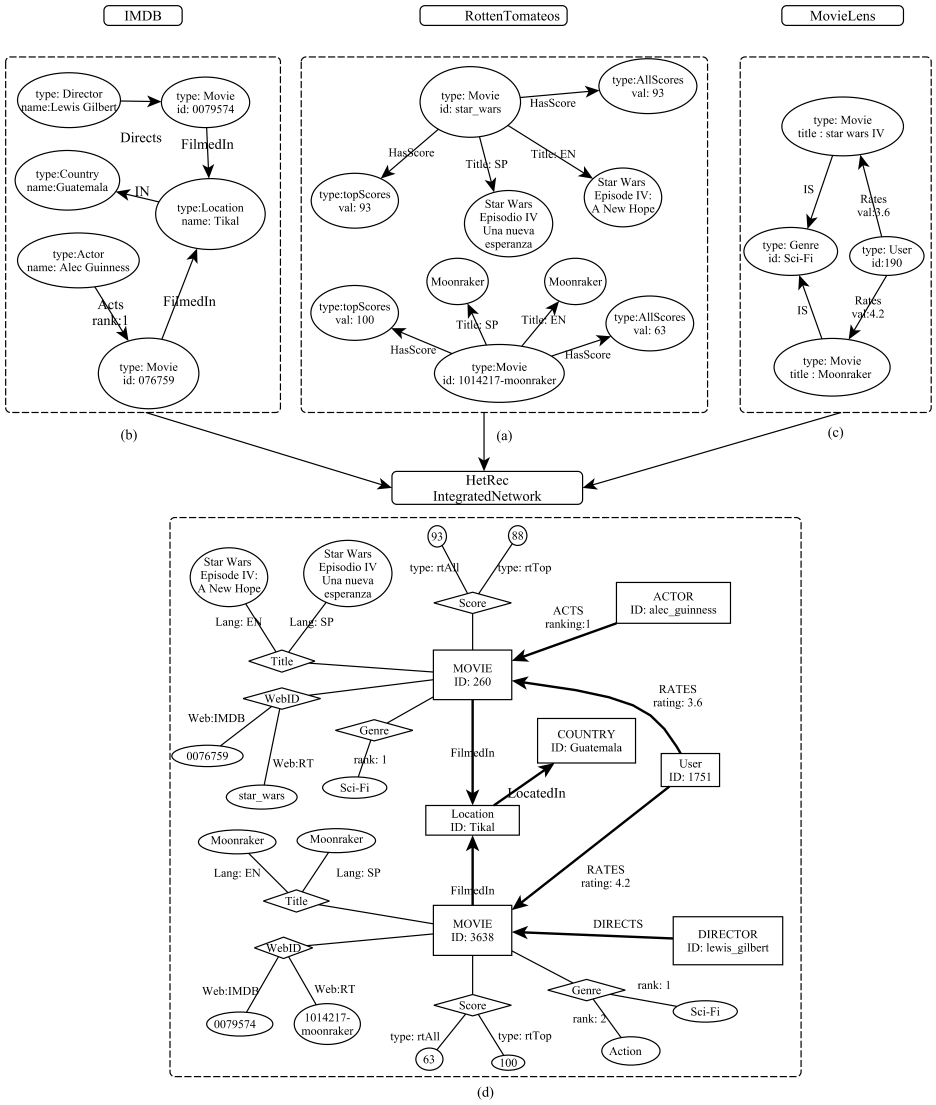


Fig. 3.2: Integration of Property Graph with GRAD

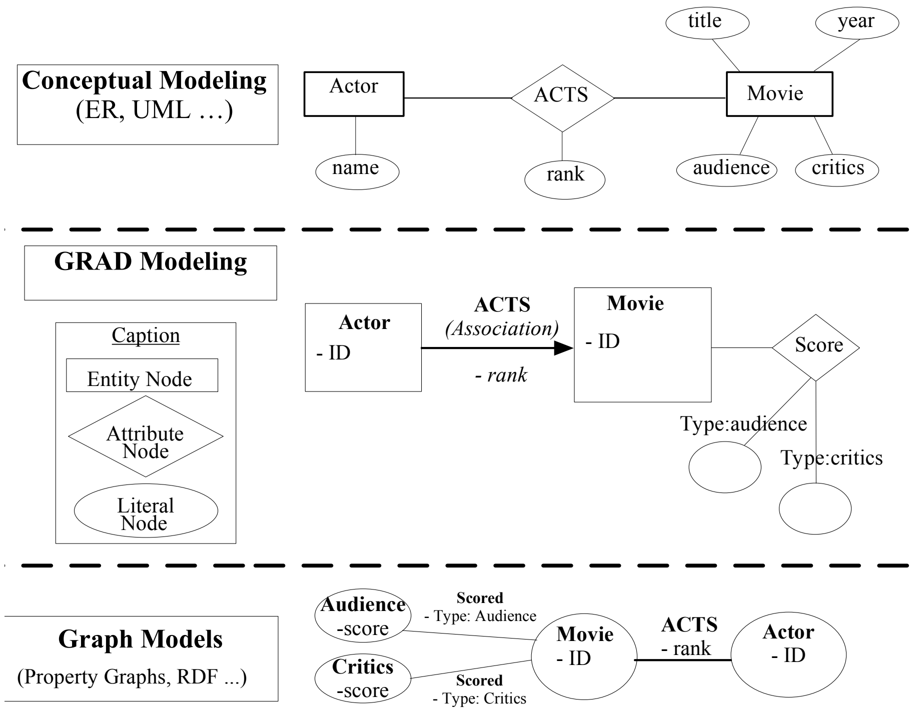


Fig. 3.3: Graph Modeling with GRAD

these conceptual models, a language is defined. Currently, the most used languages are the Unified Modeling Language (UML), and the Extended Entity-Relationship Model (EER). A study by Keet et al. [75] shows that these languages have some common core entities. For example, object type, relationship, and attribute are concepts present in most current modeling languages, although naming conventions might differ.
Of particular interest for this thesis is the topic of graph data modeling. Indeed, graphs provide a flexible and rich toolkit for knowledge representation. Graph-based data models are more generic than common representations such as relational tables. Indeed, many widely-used models are represented using graph structures. For example, the entity-relationship model [31] is represented as a set of nodes describing entities related with edges representing different relationships between them. Ontologies are also represented as graphs of inter-related primitive entities from the same domain. XML files are structured as trees, which are also specific graphs focused on the hierarchical model rather than the networked model.

In this chapter, we introduce GRAD, a new graph data model inspired by common modeling languages and designed for graph warehousing. We map the main concepts from modeling languages to GRAD, and formally define them using the graph data structures. Figure 3.3 illustrates the positioning of GRAD as an abstraction that maps conceptual modeling languages such as ER to a common graph model such as property graph. The mapping is illustrated using the example of the movie-actor network adopted in this chapter. It shows GRAD's ability to capture traditional modeling concepts and to project them on an existing model. GRAD extends property graphs by providing an advanced abstraction for graph analytics while remaining generic enough to model different domains. It introduces specific annotations and new types of nodes and edges and dictates the use of attributes and labels. Thereby, in contrast to property graphs, the design is more deterministic. From a designer's perspective, the mapping simplifies the task of representing familiar concepts from modeling languages using graphs as first-class citizens, while reducing ambiguity.

Definition 3.2. [*GRAD Graph*] *Given a finite set of labels $\mathcal{L}$, a set of property keys $\mathcal{K}$ and a set of values $\mathcal{N}$, a graph in GRAD denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H})$, is formally defined as follows:*

- *$\mathcal{V}$ is a finite set of nodes, where $\mathcal{V} = V_e \cup V_a \cup V_l$, with V_e being the set of entity nodes, V_a the set of attribute nodes, and V_l the set of literal nodes;*
- *$\mathcal{E}$ is a finite set of edges, where $\mathcal{E} = E_e \cup E_a \cup E_l$, with V_e being the set of entity edges, E_a the set of attribute edges, and E_l the set of literal edges;*
- *$\rho : \mathcal{E} \to \mathcal{V} \times \mathcal{V}$ is the function that maps each edge to its pair of nodes, with $\rho(e) = (u, v)$ denoting a directed edge from u to v;*
- *$\mathcal{F} : V_e \cup V_a \cup E_e \longrightarrow \mathcal{L}$ is the function that assigns to each entity node, attribute node, and entity edge a label;*
- *$\mathcal{H} : (V_e \cup V_l \cup E_e \cup E_l) \times \mathcal{K} \to \mathcal{N}$, is a partial function assigning to a node (resp. edge) the value of its attribute, such that, for a node (resp. edge) $v \in \mathcal{V}$, the function $\mathcal{H}(v, k_i)$ returns the value $x \in \mathcal{N}$ of its $i-th$ attribute identified by the key $k_i \in \mathcal{K}$.*

Figure 3.4 depicts the graphical notation of the different types of nodes and edges introduced by GRAD, and how they are related. In what follows, we formally define each of the GRAD data structures.

The first key concept, present in modeling languages, that we adopt on GRAD is the concept of class. In modeling languages, a class represents a set of things of a specific kind that share a common structure and relationships. A class is characterized by a predicate, such that all elements belonging to the same class satisfy the class's predicate. For the sake of simplicity, we limit

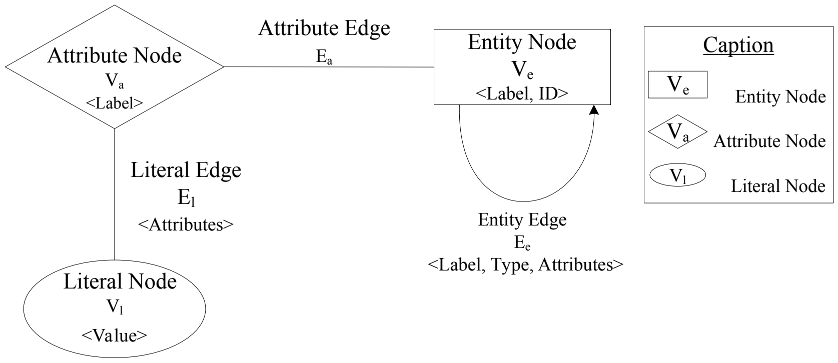


Fig. 3.4: Graphical Notation of GRAD Structures

the discussion to unary predicates, although more complex predicates could be defined and applied as well. Formally, we define a class of graph elements as follows:

Definition 3.3. [*Class*] *A class $\sum$ in GRAD describes a set of graph elements that satisfy a unary predicate applied on the labels of entity nodes. Each class Σ_i is characterized by a label C_i. Therefore, $\forall v \in V_e$, where v belongs to the class Σ_i, characterized by the label C_i, iff $C_i = \mathcal{F}(v)$.*

The set of k classes on a graph is disjoint (i.e., $\forall i, j, \Sigma_i \cap \Sigma_j = \emptyset$), and is denoted as $\sum = \{\Sigma_1, \Sigma_2, ..., \Sigma_n\}$. $\mathcal{C} = \{C_1, C_2, ..., C_n\}$ is the set of all labels characterizing classes on $\sum$, with $\mathcal{C} \subseteq \mathcal{L}$.

Example 3.1
Classes in MovieLens could be $\mathcal{C} = \{MOVIE, ACTOR, USER\}$. The predicate of the class describing movies is expressed as: $\mathcal{F}(v) = "MOVIE"$, where v is a movie node.

The second key concept from modeling languages that we must represent using GRAD is the notion of object. Objects describe concrete concepts (i.e., real-world entities) using unique identifiers, a set of attributes describing their state, and a set of relationships. Each object belongs to only one class at a time, and objects satisfying the same predicate are grouped in the same class. In GRAD, the core of an object is represented by the entity node which contains the label and the identifier attributes of the real-world entity. Attribute nodes are attached to entity nodes and denote non-identifier attributes of the object, and might be multi-valued. Literal nodes record the actual value of their corresponding attribute node each time a new value is added. We formally define each of these concepts as follows:

Definition 3.4. [*Entity Node*] *An entity node $v \in V_e$ represents the core information about a real-world entity. It contains the "type" of the entity (i.e., the class to which it belongs) represented by the label $\mathcal{F}(v)$, and its set of identifiers ID_i that unequivocally identify the entity node among the nodes of the same class. Each element of the set ID_i is immutable and might be simple or composite. Each entity node represents the core part of the information about the real-world graph element. The concept of node identifier maps to the concept of the primary key in databases and is considered to be domain-related and not system-generated.*

Example 3.2
An entity node of the class $MOVIE$ could be represented with a simple identifier (e.g., $\langle MOVIE, \{3638\}\rangle$) or a tuple (e.g., $\langle MOVIE, \{3638, StarWarsIV\}\rangle$) where the movie identifier is composed of the identifiers on the source websites IMDB and Rotten Tomatoes.

Attributes of real-world entities might have different values according to context change or new data integration procedures (e.g., updates, timestamping, etc.). We capture these changes using attribute and literal nodes, defined next.

Definition 3.5. [*Attribute Node*] *An attribute node $v_i \in V_a$ denotes an attribute of its corresponding entity node that is not an identifier and might have multiple values. An attribute node should be associated with exactly one entity node, and is defined by its label $l_i = \mathcal{F}(v_j)$ that represents the attribute's name. $\mathcal{L}_a \subset \mathcal{L}$ denotes the set of labels of attribute nodes.*

Example 3.3
In the MovieLens network, an attribute node associated with a movie entity node may represent its rating, title translated in multiple languages, or identifiers on each website. In that case, the entity nodes of the class *Movie* will have a set of associated attribute nodes labeled *Score*, *Title*, and *WebID*.

The actual value of an attribute is represented using literal nodes. GRAD captures the concept of a multi-valued attribute (as defined on typical EER models) using an attribute node with all its related literal nodes.

Definition 3.6. [*Literal Node*] *A literal node $v \in V_l$ represents the value of its corresponding attribute node ($u \in V_a$) in a given context (defined by the literal* *edge). This value could be simple or composite. The function $\mathcal{H}(v, \mathcal{F}(u))$ returns the actual value of u stored in v.*

Example 3.4
An attribute node labeled "title" is attached to movie nodes to represent the movie title in different languages. The actual new title is added as a literal node (e.g., the title of the movie translated from English to French).

Given the node types presented above, we study the types of relationships that could link them. In conceptual modeling languages (e.g., UML) various kinds of relationships between classifiers are supported, such as (1) Association, (2) Generalization, and (3) Dependency. Aggregation (representing part-whole relationship) and Composition (representing ownership relationships, with lifetime dependency) are treated as specific associations, while Realization and Usage are sub-types of Dependency [113]. Similarly to YAM^2 [2], GRAD is focused on data modeling, therefore we do not consider dependency relationships since they are mainly designed for application modeling purposes. Moreover, in this chapter we consider only binary relationships between pairs of nodes, although hypergraphs could be used to model edges connecting an arbitrary number of nodes [18]. Regarding the nodes they link, we classify edges as entity, attribute, and literal edges, defined next.

Definition 3.7. [*Entity Edge*] *An entity edge $e \in E_e$ between a pair of entity nodes $v_s, v_e \in V_e$ is defined as: $e = (v_s, v_e, type)$, where $\rho(e) = (v_s, v_e)$, $l = \mathcal{F}(e)$ is the entity edge's label, $\mathcal{H}(e, k_i)$ returns the $i-th$ attribute of e, and type denotes the specific type of the relationship, with $type \in \{Association, Generalization, Aggregation, Composition\}$. $\mathcal{L}_e \subset \mathcal{L}$ denotes the set of labels of entity edges.*

To allow richer semantics to be expressed on the graph, we define four types of entity edges as follows:

- Association edge: This is the most common type of relationship, where two entity nodes are associated with each other. Association edges have the same meaning as the association concept in UML. The set of association edges is denoted E_{as}.
- Generalization edge: This relationship relates a subclass entity node to its superclass entity node. It is usually referred to as an "Is A" relationship, and maps to generalization in UML. Each entity node can be generalized to at most one superclass (i.e., have at most one outgoing generalization edge from the subclass to the superclass). The set of composition edges is denoted E_g.

- Aggregation edge: Describes a whole/part relationship between two entity nodes. In UML this maps to the concept of (shared) aggregation and reflects the weak dependency between the two entities. Each entity node could be part of at most one whole element (i.e., have at most one outgoing aggregation edge from the part to the whole). This type of edges describes a hierarchical relationship between data entities, such as the relationship between cities and countries. The set of aggregation edges is denoted as E_{ag}.
- Composition edge: This is a stronger form of aggregation that reflects that the existence of the part depends on the existence of the composite (whole) entity node. Therefore, if the node representing the whole is deleted, the part is consequently deleted. We describe the entity node representing the part as a weak entity node that can only exist if its composite exists. Each entity node could be included as a part of at most one composite (i.e., have at most one outgoing composition edge from the part to the whole). This relationship maps to the composition (composite aggregation) in UML. The set of composition edges is denoted as E_c.

An entity edge must be typed using one of the four types defined above: $E_e = E_{as} \cup E_g \cup E_{ag} \cup E_c$. Moreover, each entity node might have many association edges, but at most one outgoing generalization (similarly aggregation or composition) edge. We use the many-to-one property of these relationships (i.e., generalization, aggregation, and composition) to introduce the notion of the parent-child relationship from the part to the whole, and from the subclass to the superclass.

Example 3.5
As an example of association edges, the relationship between movies and actors is represented by an entity edge labeled "ACTS" and having the attribute ranking to denote the actor's ranking in the movie. For generalization edges, if we imagine adding the class "crew of a movie" to the network, then both actors and directors belong to the "Crew" class. Finally, the relationship between cities and countries is a composition, because whenever a country is removed, all of its cities are removed as well.

In addition to entity edges, we introduce two other types of edges where the end nodes could be an attribute or a literal node. Attribute edges keep track of the changing attributes extracted as new nodes and are defined next.

Definition 3.8. [*Attribute Edge*] *An attribute edge represents a composition relationship (i.e., a life-cycle dependency where the same properties that were introduced* *earlier for composition relationships are valid for this edge also) between an attribute node and its parent entity node. Attribute edges do not embed attributes or labels. The set of attribute edges is denoted as $E_a \subseteq V_e \times V_a$.*

Definition 3.9. [*Literal Edge*] *A literal edge reflects a composition relationship between a literal node (part) and its parent attribute node (composite). Each literal edges has a set of attributes. Each attribute describes part of the context for the value on the literal node. The set of literal edges is denoted as $E_l \subseteq V_a \times V_l$.*

Example 3.6
A literal edge might indicate the type of the score given to a movie, while the score value is stored in the related literal node.

Following our definition of composition, we describe attribute (resp. literal) nodes as weak nodes (i.e., representing weak entities) dependent on the entity (resp. attribute) node (representing the strong entity) to which they are linked by their attribute (resp. literal) edge. We also preserve the general notation to minimally denote an edge e directed from a node v_a to a node v_b as: $e = (v_a, v_b)$.

We use the concept of hypernode to fully represent a real-world object. Hypernodes are useful for representing various concepts such as aggregation or encapsulation of graph elements, providing a coarser view of the graph as it enables a higher-level design and analysis of the network. We represent hypernodes in GRAD using subgraphs. In graph theory, G' is called a subgraph of G if G' contains a subset of nodes V' and a subset of edges E' of G. If G' preserves all the edges originally present in G between the subset of nodes V', then the subgraph is called induced, otherwise it is called a partial subgraph. Therefore, a hypernode in GRAD is an induced subgraph of the data graph. It is represented as a two-level tree where the entity node is the root and the literal nodes are the leaves. Each hypernode groups an entity node, all its attribute, and literal nodes, and the edges between them. The concept of hypernode is a key differentiator of GRAD from other models, yet it could be mapped to named graphs in RDF. With the graph data structures formally defined above, we define a GRAD hypernode as follows:

Definition 3.10. [*GRAD Hypernode*] *A hypernode in GRAD (whose entity node is $v_i \in V_e$) is defined formally as a subgraph $\Gamma_{v_i} = (V, E)$, where $V = \{v_i\} \cup V_{a_i} \cup V_{l_i}$ and $E = E_{a_i} \cup E_{l_i}$. $V_{a_i} \subseteq V_a$ is the set of all the attribute nodes attached to v_i, and $V_{l_i} \subseteq V_l$ is the set of all the literal nodes attached to the attribute nodes of V_{a_i}. Similarly, $E_{a_i} \subseteq E_a$ (resp., $E_{l_i} \subseteq E_l$) are the edges linking attribute nodes from V_{a_i} to v_i (resp., linking nodes form V_{l_i} to their attribute nodes from V_{a_i}). The label of an entity node defines the class to which its encapsulating hypernode belongs.*

Example 3.7
Each movie in *MovieLens* is represented by a hypernode that contains the entity node representing the movie and its rating as an attribute node. The rating keeps track of the different values of the movie score given by each community.

Note that each node, attribute edge and literal edge, are part of only one hypernode: $\forall u \in \mathcal{V}$ (resp. , $e \in E_a \cup E_l$), $\exists! \Gamma_v \mid u \in \Gamma_v$ (resp., $e \in \Gamma_v$). All data on the graph belonging to a given real-world entity are gathered and integrated into one hypernode, which simplifies the data integration tasks to be performed later. Moreover, the set of labels of a GRAD graph is the union of the disjoint sets of entity nodes, entity edges and attribute nodes: $\mathcal{L} = \mathcal{C} \cup \mathcal{L}_e \cup \mathcal{L}_a$. Although GRAD puts a focus on the analysis of node contents using the hypernode structure (and the node attribute reification mechanism), edge content could receive a similar focus by simply transforming edges to nodes then applying GRAD.

UML is a well-defined and accepted standard that gained wide adoption in the modeling and database community. For a more standard representation, we introduce the UML class diagram of the data model in Figure 3.5.

Example 3.8 (MovieLens with GRAD notations)
The subgraph of the MovieLens network depicted in Figure 3.6 is represented using GRAD structures as follows:

- The classes are: $\mathcal{C} = \{MOVIE, LOCATION, COUNTRY, ACOTR, DIRECTOR\}$.
- The label of the attribute node is: $\mathcal{L}_a = \{Rating\}$.
- The nodes are: $V_e = \{v_1, v_2, v_3, v_4, v_5, v_6\}, V_a = \{v_{Rating}\}$ and $V_l = \{v_{critics}, v_{audience}\}$.
- The label of nodes are: $\mathcal{F}(v_1) = "MOVIE"$, $\mathcal{F}(v_2) = "LOCATION"$, $\mathcal{F}(v_3) = "COUNTRY"$, $\mathcal{F}(v_4) = "DIRECTOR"$, $\mathcal{F}(v_5) = \mathcal{F}(v_6) = "ACTOR"$, $\mathcal{F}(v_{Rating}) = "Rating"$.
- The edges are: $E_{as} = \{e_{12}, e_{14}, e_{15}, e_{16}\}, E_c = \{e_{23}\}, E_a = \{e_{1r}\}$, and $E_l = \{e_{ra}, e_{rc}\}$.
- The labels of the entity edges are: $\mathcal{L}_e = \{ACTS, DIRECTS, Filmed\_IN, Located\_IN\}$, with $\mathcal{F}(e_{12}) = "Filmed\_IN", \mathcal{F}(e_{23}) = "Located\_IN", \mathcal{F}(e_{14}) = "DIRECTS"$, and $\mathcal{F}(e_{15}) = \mathcal{F}(e_{16}) = "ACTS"$.

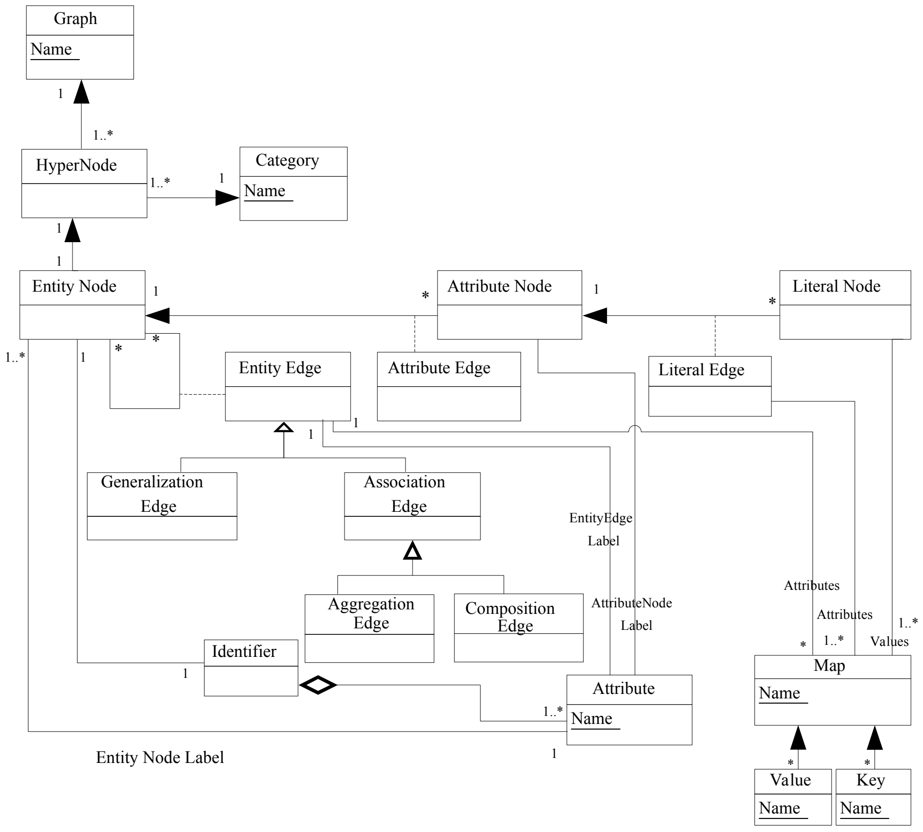


Fig. 3.5: GRAD Modeling with UML

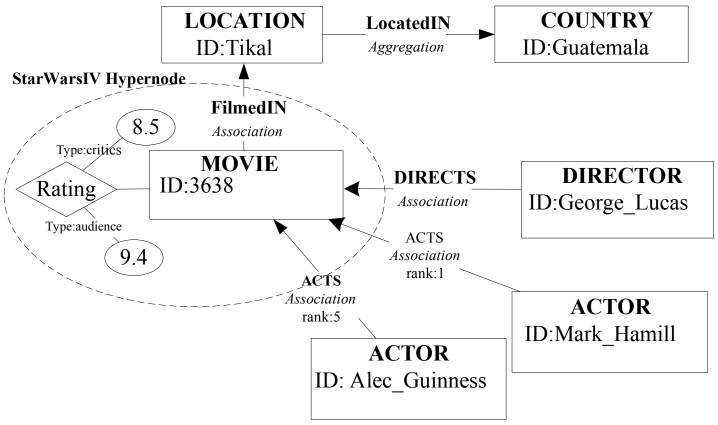


Fig. 3.6: MovieLens with GRAD Notations

- The attributes of entity nodes are: $\mathcal{H}(v_1, ID) = \{3638\}, \mathcal{H}(v_1, title) = \{Star\ \ Wars\ \ IV\}$, $\mathcal{H}(v_2, ID) = \{Tikal\}$, $\mathcal{H}(v_3, ID) = \{Guatemala\}$, $\mathcal{H}(v_4, ID) = \{George\_Lucas\}, \mathcal{H}(v_5, ID) = \{Marc\_Hamill\}, \mathcal{H}(v_6, ID) = \{Alec\_Guinness\}$.

- The values of literal nodes are: $\mathcal{H}(v_{critics}, "Rating") = 8.5$, $\mathcal{H}(v_{audience}, "Rating") = 9.4$

- The attributes of entity edges are: $\mathcal{H}(e_{15}, rank) = 1, \mathcal{H}(e_{16}, rank) = 5$

- The attributes of literal edges are: $\mathcal{H}(e_{rc}, Type) = "critics", \mathcal{H}(e_{ra}, Type) = "audience"$

4 Integrity Constraints

Integrity constraints are the general rules describing the consistent database states, or change of states, or both [34]. They play a fundamental role in data quality enforcement within data management systems, and especially for decision support systems. Integrity constraints were first introduced with the relational model and later studied in object-oriented and semi-structured data models, and lately in graph models such as in [101] or the SHACL specification for RDF. This topic is also present in the context of graph databases, where the efforts were centered around applying integrity constraint concepts from the relational model on graph data [6]. In this section, we provide a formal definition for the integrity rules applied over the graph structures presented in the previous section. As discussed in Angles et al. [10], several integrity constraints were proposed for graph databases in the literature. We identify the ones relevant to GRAD as follows:

- Graph entity integrity: This constraint is used to guarantee that each real-world entity is represented by a unique hypernode. It also provides the mechanisms for nodes and edges identification through specific attributes (i.e., ID) and/or structural properties such as neighborhoods.
- Semantic constraints: These are user-defined constraints. The first type represents the assertions (i.e., topological and value-based constraints) users wish to define for the graph elements. The second type focuses on cardinality checking between classes of nodes. We now study both kinds of constraints in more detail.

4.1 Graph Entity Integrity

This first category of integrity constraints aims at guaranteeing that a real-world entity is represented only once in the graph database. This prevents data redundancy of data and helps to fulfill consistent update and deletion of graph entities. Since we are managing labeled graphs, we need to ensure a set of constraints on the labeling of graph elements. These constraints are defined as follows:

- Each class should have exactly one unique label different from all the labels of the other classes: $\forall \Sigma_i, \Sigma_j \in \sum$, if $\Sigma_i \neq \Sigma_j$ then $C_i \neq C_j$.
- Entity edge labels between different classes are unique: let $v_i, v_j, v_k \in V_e$, $\mathcal{F}(v_i, v_j) = \mathcal{F}(v_i, v_k)$ iff $\mathcal{F}(v_j) = \mathcal{F}(v_k)$.
- Two entity edges with the same label cannot link the same pair of entity nodes: let $v_i, v_j \in V_e$ and $l_1, l_2 \in \mathcal{L}_e$, if $e_m = (v_i, v_j, type_m)$ and $e_n = (v_i, v_j, type_n)$ then $l_1 \neq l_2$.

Entities in the graph are identified by their unique identifier, their neighborhood, or both, using the following identification mechanisms:

- Each entity node should have an identifier that is unique within the class of that node. Therefore each entity node v_i is uniquely identified by the pair composed of its identifier ID_i and its label C_i represented by $\langle C_i, ID_i \rangle$. The exception here is on the identification of weak entity nodes (i.e., entity nodes related by a composition relationship to another composite entity node). These weak entity nodes require also the identifier of their parent entity node to be identified. Let ID_j be the identifier of a weak entity node v_j, whose parent is identified by ID_{parent}, then v_j is identified by the triple $\langle C_j, ID_{parent}, ID_j \rangle$.

- GRAD supports multigraphs. Thus to unequivocally identify an entity edge, the identifiers of the nodes it links are not enough, we also need the edge's label. We previously stated that two entity edges with the same label cannot link the same pair of entity nodes. Therefore, each entity edge e_i is uniquely identified by the triple comprised of its label $l_i \in \mathcal{L}_e$ and the identifiers (ID_j, ID_k) of the entity nodes it links (v_j and v_k) represented by $\langle l_i, ID_j, ID_k \rangle$.

- An attribute node $v_i \in V_a$, associated to an entity node $u \in V_e$, is identified by the pair $\langle l_i, ID_j \rangle$ comprised of its label $l_i \in \mathcal{L}_a$ and the identifier ID_j of u. An attribute edge is identified by the entity node and attribute node it links.

4.2 Semantic Constraints

Based on the knowledge of a specific domain, end-users might need to make some semantic restrictions over the graph data. The goal of these constraints is to guarantee the compliance of the graph data with respect to domain-specific rules. Since these constraints could not be automatically identified and captured by the system, they need to be explicitly expressed by users. In GRAD, we also choose them to be represented using graph patterns. Thus, before defining the notion of semantic constraints, we first introduce the concept of graph pattern in GRAD. A graph pattern P is a predicate on the topology (specifying conditions on the structural properties of the graph) and attributes (specifying conditions on the values of the attributes) of the graph elements. The topic of graph pattern matching is well-studied in the graph theory literature [44], we formally define a graph pattern in GRAD as follows:

Definition 3.11. [***Graph Pattern***] *A pattern graph is defined as* $\mathcal{P} = (V_P, E_P, \alpha, \beta)$*, where:*

- *V_P is a set of nodes and E_P is a set of edges;*

- *α is a function defined on $V_P \cup E_P$ such that for each node $u \in V_P$ (resp., edge $e \in E_P$), $\alpha(u)$ (resp., $\alpha(e)$) is the predicate applied on the label of u (resp., e). This predicate compares the label $l_i = \mathcal{F}(u)$ of u (resp., of e) with a string s_j. The comparison is of the form $l_i \; op \; s_j$, and is performed using one of the two equality comparison operators $=, \neq$;*

- *β : is a function defined on $V_P \cup E_P$ such that for each node $u \in V_P$ (resp., edge $e \in E_P$) , $\beta(u)$ (resp., $\beta(e)$) is the predicate applied on the attributes of u (resp.,* *e). This predicate is a conjunction of atomic formulas each of which compares a constant c with the value x of an attribute of u (resp., e) using a given operator op_i. The comparison is performed using any of the following operators: $<, \leq, =, \neq, >, \geq$. Hence, $\beta(u)$ (resp., $\beta(e)$) is a conjunction of comparisons of the form: $c\ op_i\ x$.*

We represent each atomic formula of the predicate functions (*i.e.*, α, β) as $f_{i(op_i, c_j)}(val) \rightarrow \{true, false\}$. The predicate function is evaluated to true if all its atomic formulas are true. In this chapter, we focus on the case of the conjunction between predicates. However, the same approach could be extended to support disjunctions or an arbitrary combination of conjunctions and disjunction of predicates.

In what follows, we use subgraph isomorphism to identify all the subgraphs that match a given graph pattern. We say that a subgraph G' of a given graph G matches a query pattern $\mathcal{P}$ by isomorphism, and we denote this as $G' \cong \mathcal{P}$.

Definition 3.12. [*Subgraph Matching*] *Consider a GRAD graph $\mathcal{G}$ and a subgraph of $\mathcal{G}$, denoted $G' = (V', E', \rho, \mathcal{F}, \mathcal{H})$, matches a pattern P using graph isomorphism, if there is a bijective function $\Phi : V_P \rightarrow V'$ such that:*

- *Every node on V_p has an image node on V' by the bijective function Φ, and every edge from E_p has an image edge on E'. Formally, $\forall u, u' \in V_p$, $e = (u, u') \in E_p$, iff $\Phi(u), \Phi(u') \in V'$ and $(\Phi(u), \Phi(u')) \in E'$;*
- *$\forall u \in V'_e \cup V'_a, \alpha(u)$ holds, i.e., $u \in V'_e \cup V'_a$ iff $\exists l_i \in \mathcal{L}$ such that $\mathcal{F}(u) \models \alpha(u)$;*
- *$\forall e \in E'_e, \alpha(e)$ holds, i.e., $e \in E'_e$ iff $\exists l_j \in \mathcal{L}_e$ such that $\mathcal{F}(e) \models \alpha(e)$;*
- *$\forall u \in V'_e$, $\beta(u)$ holds, i.e., $u \in V'$ iff $\exists k_i \in \mathcal{K}$ such that $\mathcal{H}(u, k_i) \models \beta(u)$;*
- *$\forall u \in V'_l$, where its attribute node isv_a, $\beta(u)$ holds, i.e., $u \in V'_l$ iff $\mathcal{H}(u, \mathcal{F}(v_a)) \models \beta(u)$;*
- *$\forall e \in E'_e \cup E'_l, \beta(e)$ holds, i.e., $e \in E'_e \cup E'_l$ iff $\exists k_j \in \mathcal{K}$ such that $\mathcal{H}(e, k_j) \models \beta(e)$.*

It should be noted that the bijective function Φ specifies the minimum set of properties a graph element should have. For a pair of nodes $v \in V_p, v_x \in V', v_x = \Phi(v)$ means that (1) v_x has at least all the attributes of v for which the predicates of P hold, and (2) v_x has at least all the relationships v has. However, the matched nodes $v_x \in V'$ could have more relationships and attributes than those specified on the pattern's description.

The problem of graph matching using subgraph isomorphism is known to be NP-complete. Indexing techniques were proposed in the literature to accelerate matching algorithms. For some domains and applications, weaker forms of graph matching are sufficient. Hence, the strict constraint of isomorphism is relaxed, reducing the complexity of the matching operation. A set of alternatives was introduced and discussed by Fan et al. [44], such as graph pattern matching by simulation, dual simulation, and strong simulation.

We now address the semantic constraints. In particular, we study here two categories of such constraints: assertions and multiplicities.

Assertions

These are predicates applied on the graph data and that must always be satisfied. In GRAD, assertions are represented using graph patterns to specify the topological and content-based constraints chosen by the user.

Definition 3.13. [*Assertion*] *An assertion is represented by a graph pattern P. A graph G is said to be compliant to the assertion if for every entity node ($v \in V_e$) matched by a node on the assertion pattern ($v_i \in V_p$), there exist a subgraph $G' \subseteq G$ such that G' matches P.*
Formally: $\forall v_e \in V_e, v_i \in P \ \mid \Phi(v_i) = v_e \Rightarrow \exists!\ G' \subseteq G \mid v_e \in G'$ *and* $G' \cong P$.

Example 4.1 (Assertion on Movies)
For example, a user can state that all movies on the graph should have an attribute node labeled "Rating", with the rating value by "Audience" being above 7. The movie has to be related to a set of actors and a director. If these conditions are not met, the insertion transaction of the movie in the graph should fail. The pattern predicate is represented graphically in Figure 3.7 (i.e., as a graph whose structure represents the topological constraints and the values on its attributes represent the conditions on the output attributes). This example pattern is represented in GRAD as follows:

- Set of nodes: $V_e = \{v_A, v_M, v_D\}$, $V_a = \{v_r\}$ and $V_l = \{v_l\}$.
- Set of edges and their respective end-nodes: $E_e = \{e_D, e_A\}$, with $e_D = (v_D, v_M)$ and $e_A = (v_A, v_M)$. $E_a = \{e_r = (v_M, v_r)\}$ and $E_l = \{e_t = (v_r, v_l)\}$.
- Predicates on the labels of entity nodes: $\alpha(v_A) : \mathcal{F}(v_A) = "ACTOR"$, $\alpha(v_M) : \mathcal{F}(v_M) = "MOVIE"$ and $\alpha(v_D) : \mathcal{F}(v_D) = "DIRECTOR"$.
- Predicate on the label of the attribute nodes: $\alpha(v_r) : \mathcal{F}(v_r) = "Rating"$.
- Predicates on the labels of entity edges: $\alpha(e_A) : \mathcal{F}(e_A) = "ACTS"$ and $\alpha(e_D) : \mathcal{F}(e_D) = "DIRECTS"$.

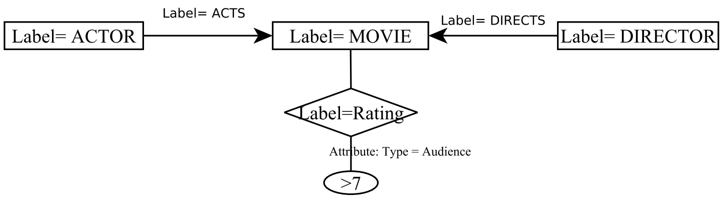


Fig. 3.7: Assertion Pattern on Movie Nodes

- Predicate on the attribute of literal edges: $\beta(e_t) : \mathcal{H}(e_t, Type) = "Audience"$.
- Predicate on the value of literal nodes: $\beta(v_l) : \mathcal{H}(v_l, "Rating") > 7$.

Multiplicities

They are applied between classes of entity nodes to define the number of relationships a node from a given class can have with nodes from other classes through a particular entity edge. This is a user-defined constraint that is well-known in conceptual modeling languages. In GRAD, multiplicities specify the minimum and the maximum number of relationships an entity node is allowed to have with entity nodes from other classes, through entity edges of a given label. If not set by the user, the default multiplicity is unbounded.

Definition 3.14. [*Multiplicity*] *Formally, this constraint is represented by a function μ defined on $\Sigma \times \mathcal{L}_e \times \Sigma$. For a given pair of entity node classes ($\Sigma_i, \Sigma_j \in \sum$), and an edge's label ($l_k \in \mathcal{L}_e$), $\mu(\Sigma_i, l_k, \Sigma_j)$ defines the multiplicity predicate. It specifies the maximum and minimum number of entity edges labeled l_k that could exist between a given node $v_i \in \Sigma_i$ and the set of nodes $v_j \in \Sigma_j$, and vice versa. Note that the multiplicity predicate is a conjunction of at most two atomic formulas (specifying the lower and upper bounds) at each endpoint. Each predicate compares the number of edges with the given maximum and/or minimum using a given comparison operator from the following list: $<, \leq, =, \geq, >, \neq$.*

Example 4.2 (Multiplicity of Movies)
A user may want to state that an actor can participate in many movies but that a movie should have at least one actor. In the first case, the maximum range is unbounded, while the second range is greater than one. We represent this constraint between actors and movies related by the $ACTS$ relationship using UML notations as $[1..*, *]$. Figure 3.8 illustrates this example of constraints using a graph-based representation. Note that the multiplicities should not violate the specific properties of entity edges (e.g., to-one relationships between nodes related by composition).

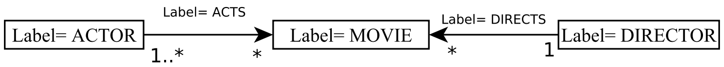


Fig. 3.8: Multiplicity of Movie Nodes

While multiplicities focus solely on cardinality checking between classes of nodes, assertions extend the constraints' scope to target the topology and the content allowed on the graph. Naturally, the two constraints could be combined by adding for example the function μ to the assertion's pattern $\mathcal{P}$. In such a case, the assertions will define the constraints on the topology and attributes of graph elements, and the multiplicity defines the allowed number of relationships between the different entity nodes.

Many other integrity constraints were studied in the literature of graph database models. For example, functional dependencies and referential integrity were considered by some graph database models. Functional dependencies describe the fact that the values of an element's attribute determine the values of another one. Referential integrity, on the other hand, as defined on the relational model, guarantees that only existing entities are referenced. These two constraints are designed to put constraints on implicit relationships existing between data entities. However, these relationships are explicitly expressed on the graph using edges. We can therefore use specific graph edges (e.g., composition) to reflect the same semantic. Moreover, as acknowledged in [10], these concepts are inherited from the relational model and remain difficult to project directly on the graph database models.

5 Graph Algebra

In this section, we complete the GRAD database model with a graph algebra, consisting of a set of algebraic operators specific for online graph querying and analysis. The algebra supports the data structures of GRAD while preserving the integrity constraints. This allows users to traverse and query the network without violating the database integrity. The algebra we present in this section extends GraphQL, a graph algebra defined along the lines of the relational algebra [60]. An interesting characteristic of GraphQL is that it provides a mapping between the algebraic expressions and FLWR expressions from XQuery. The query language is declarative, graph-oriented, and suited for semi-structured data. Moreover, GraphQL is relationally complete and implements common graph querying operations such as subgraph matching, finding paths, and graph aggregations [133, 136]. A major difference between the original GraphQL and GRAD Algebra is that GRAD is oriented for the management and querying of a single large graph while GraphQL targets collections of small graphs. Single large graph analysis is usually applied on domains such as social and bibliographic networks with queries centered on pattern matching, reachability, and shortest path. The analysis of collections of graphs is usually applied on chemical and bioinformatic networks, with analysis-oriented for subgraph and supergraph queries [117].

GRAD algebra operates directly on GRAD structures, which are the fundamental construct, the operands, and the return type of all algebraic operations. By building on GraphQL, our algebra inherits the same expressive power; hence it is at least as expressive as the relational algebra. We study the closure of each algebraic operator with respect to GRAD structures and integrity constraints. When starting from a consistent database state (i.e., where the integrity constraints presented above are satisfied), the closure ensures that the integrity constraints remain satisfied after the execution of any given algebraic operation. This guarantees that each operator's output is formatted according to GRAD and thereof is a valid input for the other operators. Finally, algebraic laws regarding commutativity and associativity remain valid since the algebra is built on the lines of relational algebra. For all the operators we introduce in this section, we assume the following: The input is a GRAD graph. The general rule stating that if a node is removed, then all its edges are removed, is valid for all types of nodes. If a composite (parent) entity node is removed, all its parts (children) weak entity nodes are removed consequently. This means, for instance, that if an entity node is removed then the whole hypernode is removed, and if an attribute node is removed then all its literal nodes are removed as well. We now study the operators one by one.

5.1 Selection

The first major operation is the selection which is a subgraph extraction operation based on graph pattern matching. A selection consists in finding the subgraph (or set of subgraphs) of the data graph G that satisfies the semantic and topological constraints specified by a given graph pattern $\mathcal{P}$. The selection operates only on valid graph pattern in GRAD defined as follows:

Definition 3.15. [*Valid Pattern*] *A given graph pattern $\mathcal{P} = (V_p, E_p, \beta, \alpha)$ is a valid GRAD pattern if it satisfies the following conditions:*

- *All elements of $\mathcal{P}$ are represented using GRAD structures: $V_p \subseteq V_e \cup V_a \cup V_l$ and $E_p \subseteq E_e \cup E_a \cup E_l$. Figure 3.9-(a) shows an example of a subgraph that violates this condition. The violation here is that two attribute nodes are linked to the same literal node while in GRAD a literal node must be related to exactly a single attribute node;*
- *The definition of predicates on the content (label and attributes) of the pattern elements is optional (e.g., an entity node might not have a predicate on its label*

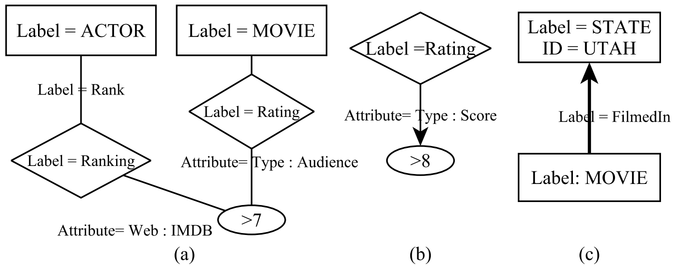


Fig. 3.9: Non-Valid GRAD Patterns

or identifier). In case no predicate is specified on an attribute or a label, all elements match the pattern. Intuitively, for a given graph element, a predicate can only be applied on the properties specified for its kind by GRAD (e.g., a predicate cannot be applied on the labels of attribute edges because by definition an attribute edge does not have a label). For example the pattern on Figure 3.9-(a) states that the attribute edge linking an actor entity node to its ranking attribute node should be labeled "Rank", while GRAD does not support labels on attribute edges;

- *The pattern does not include weak nodes without their parent nodes. That means, if the pattern includes a weak node (i.e., child nodes related by a composition relationship to their parent nodes), then the parent node should be included in the pattern as well. This is required because a weak node does not represent a complete entity and could not be identified by itself. For example, if the pattern describes an attribute node and its literal node, then it should also contain a description of their corresponding parent entity node. For example Figure 3.9-(b) is not valid because the attribute node is not associated with an entity node. Figure 3.9-(c) is not valid because entity nodes labeled "STATE" are weak entity nodes linked by composition to their parent entity nodes labeled "COUNTRY" which are not included in the pattern.*

Regarding the MovieLens working example, Figure 3.10 provides examples of valid GRAD patterns. Particularly, Figure 3.10-(a) and Figure 3.10-(b) provide a possible correction for the non-valid patterns shown in Figure 3.9-(b) and Figure 3.9-(c) respectively.

Using the formal definitions of GRAD data structures and valid GRAD pattern, we formally define the selection operator as follows:

- Input: A graph G and a valid selection pattern P.
- Mathematical notation: $\sigma_P(G)$.
- Output: A set of matched graphs $\Omega = \{(G_1, G_2, ..., G_n) \mid G_i \cong P\}$, such

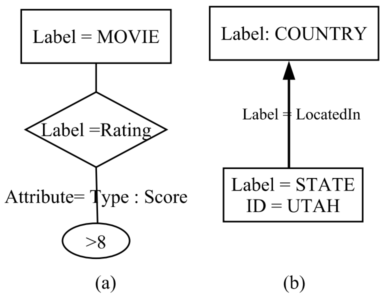


Fig. 3.10: Valid GRAD Patterns

that all G_i are subgraphs of G that match the selection pattern P by isomorphism.

- Closure condition: The closure is satisfied if the selection pattern is a valid GRAD pattern.

Example 5.1 (Top-Rated Movies Selection)
Suppose the analyst would like to extract the network of the top-rated movies by the audience (i.e., movies rated above 7 by the audience). The user is also interested in getting the top actors of each top movie (i.e., actors with a ranking equal to 1 playing in the selected top-rated movies). This query is answered by applying the following selection: $\sigma_P(G_{MovieLens})$. Here the pattern applies constraints on the labels of entity nodes ("ACTOR" and "MOVIE"), the label and attribute of their linking entity edge (the label is "ACTS", and the attribute is "Ranking = 1"), the labels of movies' attribute nodes (the label is "Rating") and the rating values (value in the literal node > 7). The input and output of this selection operation are shown in Figure 3.11.

Note that even if the pattern is not GRAD compliant, the selection could be performed to extract graph data. However, the result cannot be guaranteed to be a valid input for other operations as it violates their preconditions.

5.2 Cartesian Product

Cartesian Product is a binary operator applied to put together two collections of graphs. The algebra of this operator and is defined as follows:

- Input: Two collections of graphs S_1 and S_2.
- Mathematical notation: $S_1 \times S_2$.

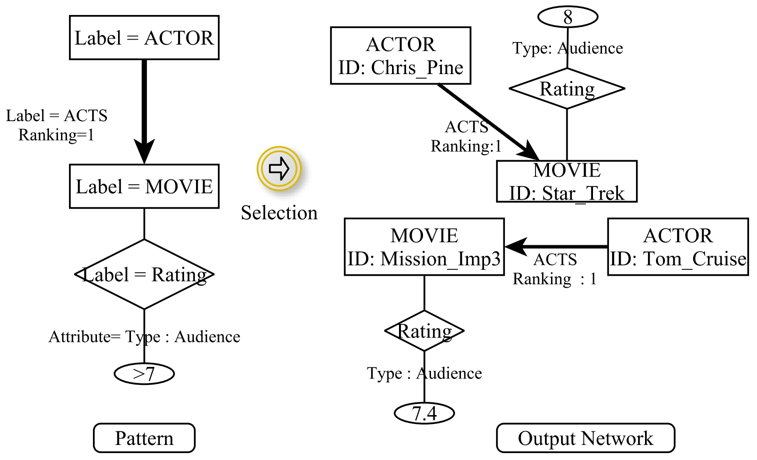


Fig. 3.11: Top Movies and Actors Network

- Output: Let $S_1 = \{G_1, G_2\}$ and $S_2 = \{G_3, G_4\}$. The result of the Cartesian product is a set of pairs of unconnected graphs. Each graph on the output is composed by a graph from S_1 and a graph from S_2 respectively: $S_1 \times S_2 = \{G_a = (G_1, G_3), G_b = (G_1, G_4), G_c = (G_2, G_3), G_d = (G_2, G_4)\}$.
- Closure condition: Cartesian product is a closed operation since it does not alter the internal structure of the hypernode neither adds new edges.

5.3 Composition

The idea of composition is to reuse the information extracted from the input graph data to generate new graphs. The composition operator is used to create a new graph based on data collected from the original graph, then formatted according to a given graph template. A graph template $\mathcal{T}$ describes the structure of the output graph by specifying the content of graph entities and their organization on the resulting graph. The template could be defined using a subgraph G'. As shown in Figure 3.12, the template describes an output network with one type of entity nodes (labeled "$ACTOR$" and with the identifier "name"). Entity nodes are related by one type of relationship (labeled "$Co-Acts$" and of type "$Association$").

Applying composition is to some extent analogous to having a function with predefined instructions (i.e., a graph template), a set of formal parameters (i.e., a graph pattern), and a set of actual parameters values (i.e., the matched graphs). First, the user needs to define the graph template that she

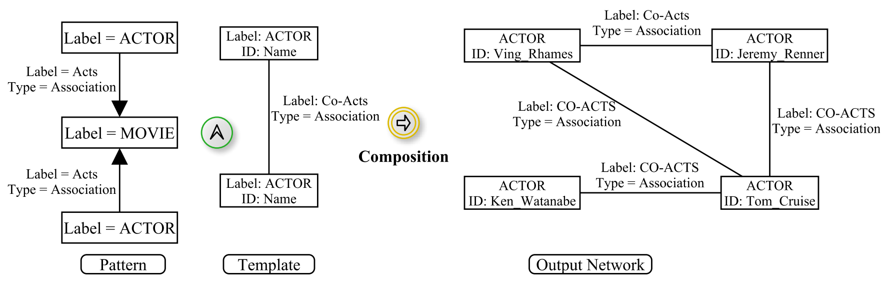


Fig. 3.12: Co-actorship Network Generation

wants to generate using data from the input graph. Then a graph pattern matching operation is applied to the input graph to retrieve a set of matched subgraphs. Finally, the composition operator instantiates the graph template using data from the matched subgraphs. The composition operator is formally defined as follows:

- Input: The initial graph G, the graph template τ, and a pattern P.
- Mathematical notation: $\omega_{\tau_P}(G)$.
- Output: Let $\Omega = \{G_1, G_2, ..., G_n\}$ be the set of n graphs matched by the pattern P such that $\Omega = \sigma_P(G)$. Let τ be the template of the output graph, and $\tau_P(G)$ be the function that instantiates the template τ. Each $G_i \in \tau_P(G)$ is then a graph that follows the template τ and filled by data matched by P from the initial graph G. The output of the composition is denoted as $\omega_{\tau_P}(G) = \{\tau_P(G_i) \mid G_i \in \Omega\}$. Hence, the result is a set of graphs instantiating the template τ_P (i.e., graph elements are created according to the template and the actual values of their attributes are assigned).
- Closure condition: The composition is closed if the pattern is valid and the template is compliant to GRAD structures and constraints.

Example 5.2 (Co-actors Composition)
Assume the analyst wants to generate the graph of co-actors from the original graph. The template is a network of actors connected by co-actorship edges. The actual names of the actor are retrieved from the input graph. The pattern matches the pairs of actor nodes that are directly connected to the same movie. This operation is depicted in Figure 3.12.

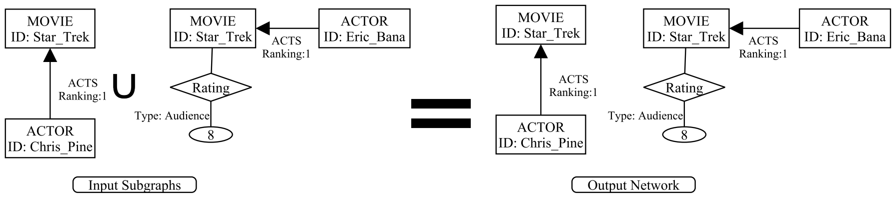


Fig. 3.13: Union of Actors of the Same Movie

5.4 Set operators: Union and Difference

The two core set operations on graphs are union and difference. A union between two graphs generates a new graph putting together the two input graphs without concatenation. Union does not introduce changes to the internal structure of each input graph. Graph union is different from its relational counterpart in that no common structure is required for executing the operation. The union operator is defined as follows:

- Input: Two initial graphs G and G′.
- Mathematical notation: $G \cup G'$.
- Output: $G'' = G \cup G'$, where all graph elements from G and G′ are on G" (i.e., $V'' = V \cup V'$, and $E'' = E \cup E'$).
- Closure condition: The union does not introduce any structural changes to the graph elements. This operation is therefore closed.

Example 5.3 (Graph Union)
While adding new graph sources to the graph database, the union operation is used simply to put together the original graph with the graph entities being added. The union is sufficient if the added entities do not exist previously on the graph. Note that as in Figure 3.13 the two subgraphs are unified but not merged. Intuitively, if an entity (e.g., a movie) is already present in the graph, an integration operation fusing the two corresponding entity node is required. We further investigate this case with the join operation introduced next.

The difference between graphs removes isomorphic subgraphs that exist in the two input graphs. The Difference operator is defined as follows:

- Input: Two graphs G and G′.
- Mathematical notation: $G - G'$.

- Output: $G - G' = G''$, where all graph elements from G isomorphic to graph elements in G′ are removed from G. While the composition is used to generate new graphs, the difference is the only operation we introduced so far to enable deletion of graph elements.
- Closure condition: The difference introduces structural changes to the graph by removing the shared subgraphs. This operation is closed as long as it respects the assumptions we introduced at the beginning of this section.

Structural Graph Join

The union or Cartesian Product operations introduced above enable putting together a collection of graphs. However, in scenarios where graph entities need to be integrated, these operations are not enough to merge redundant graph entities. We need therefore to introduce another operation, the structural graph join, or simply join, which consists in the unification of nodes and edges based on a common join predicate. The join operator is defined as follows:

- Input: Two collections of graphs S_1 and S_2, and a join predicate Pr that applies on the attributes graphs of S_1 and S_2.
- Mathematical notation: $S1 \bowtie_{Pr} S2$.
- Output: A join is performed in multiple steps. First a Cartesian Product: $S_1 \times S_2$ is applied and the result is a set of pairs of subgraphs. For this set of pairs, a selection is applied to retain only pairs of subgraphs whose elements satisfy the join predicate Pr. For each retained pair of subgraphs, their graph elements that satisfy the predicate are merged. The generation of the new graph resulting from the join is performed by the composition operation. However, this is a specific case of composition where there is no need for the pattern matching phase since the graphs to join are the input instead of the matched subgraphs. For the composition template, it consists in the union of the two joined graphs after edges and nodes unification.
- Closure condition: The join operation is closed in case no unification is required. In the case of unification of edges or nodes, the structure of the graph might change and the closure is verified at the level of the composition operation that performs the unification.

Example 5.4 (Structural Join)
Assume the analyst wants to integrate the incoming data about movies in the graph already stored in the database. Often, some movies and actors

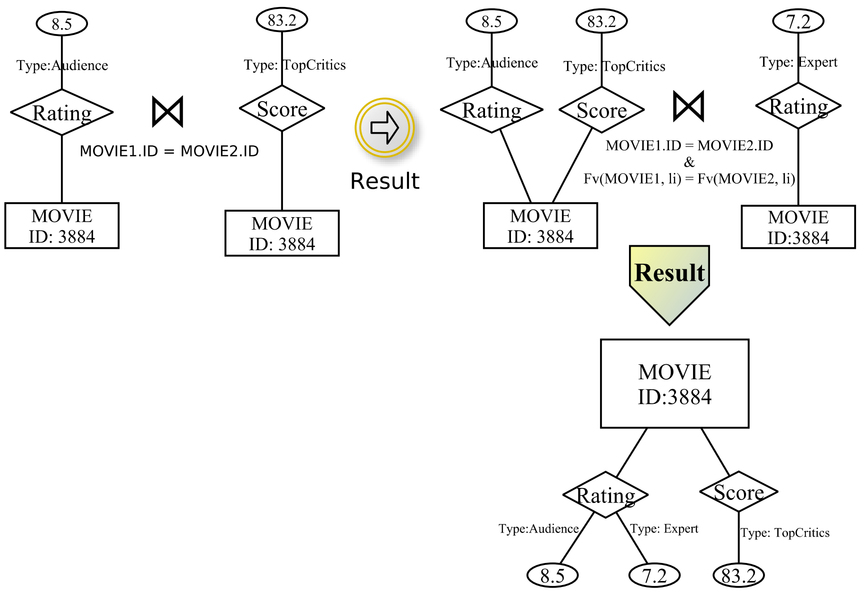


Fig. 3.14: Join on Attribute and Value Nodes

could have been already inserted in the database. These existing entities have to be merged with their corresponding incoming ones. In this example, the correspondence between the existing nodes and potential incoming nodes is performed based on the identifiers of the movie nodes. The join predicate used to match two movie hypernodes is their ID denoted as $MOVIE.ID$. After the fusion, we can notice that the two movie nodes (with ID= 3884) are merged and an additional type of attribute nodes (Score) is attached to the existing movie as shown on the first join on Figure 3.14. Once the entity nodes and their attribute nodes are merged, another possible scenario could be the fusion of attribute nodes of the movie node based on their common label l_i. This is shown in the second join in Figure 3.14, this adding the new captured rating values as literal nodes attached to the Rating attribute node.

6 Conclusion

In this chapter, we proposed a novel database model for intuitive modeling and effective querying of graph data. GRAD is a complete and native graph database model where relationships are first-class citizens. It proposes

advanced graph structures to represent real-world entities. The algebra is designed such that graphs are the operands and the output of all operators. We defined the integrity constraints to guarantee data consistency.
GRAD is a generic database model that lays the foundations for building graph management systems. It could further be extended to support specific application requirements. Extensions could be applied at the data structures, algebraic operators, or constraints to fit certain application requirements. For example, GRAD could be used for graph data warehousing and Master Data Management. It could be extended to support spatio-temporal graph databases. A further step could be the support of evolving graph databases, that takes into account both data and schema changes.

Chapter 4

A Framework for Building OLAP Cubes on Graphs

The paper has been published in the
19th East-European Conference on Advances in Databases and Information Systems (2015). The layout and content of the paper have been revised.
DOI: https://doi.org/10.1007/978-3-319-23135-8_7

Abstract

Graphs are widespread structures providing a powerful abstraction for modeling networked data. Large and complex graphs have emerged in various domains such as social networks, bioinformatics, and chemical data. However, current warehousing frameworks are not equipped to handle efficiently the multidimensional modeling and analysis of complex graph data. In this chapter, we propose a novel framework for building OLAP cubes from graph data and analyzing the graph topological properties. The framework supports the extraction and design of the candidate multidimensional spaces in property graphs. Besides property graphs, a new database model tailored for multidimensional modeling and enabling the exploration of additional candidate multidimensional spaces is introduced. We present novel techniques for OLAP aggregation of the graph and discuss the case of dimension hierarchies in graphs. Furthermore, the architecture and the implementation of our graph warehousing framework are presented and show the effectiveness of our approach.

1 Introduction

In this chapter we present our second framework bringing the value of graph analytics to decision-support systems. A major step towards extending current data warehouses with graph capabilities is to address the need for native multidimensional graph models and re-think OLAP analysis to capture and expose graph topology and content. Many approaches were proposed in the literature to extend current decision support systems with graphs [28, 135, 143]. These works suggested the first foundations for building OLAP cubes on graphs. In this literature, multidimensional concepts are embedded within the attributes of graph elements. A subset of attributes could be considered as dimensions used for aggregating the graph and performing its multi-perspective analysis. Another subset could be considered as the examined measures. However, most of the existing techniques focus on homogeneous graphs (i.e., graphs where all nodes are of the same type, and all edges are the same), while real-world graphs are complex and often heterogeneous. OLAP analysis focuses mainly on the graph topology as the measure of interest, while attributes could also provide interesting measures. The model we propose in this chapter extends the state-of-the-art to heterogeneous graphs (i.e., graphs where nodes and edges could be of different types to represent different real-world entities, and the different relationships between them). Besides, we focus on both the attributes of nodes and edges to construct further multidimensional spaces. We examine two types of spaces, inter-class and intra-class respectively. In inter-class, we focus on multidimensional structures between two types of nodes. While in intra-class, we dive into a single type of node to explore the possible multidimensional spaces that could be constructed within the same class. We extend the analysis capabilities on graphs by integrating GRAD, an analysis-oriented graph database model [50, 52]. By introducing new graph structures designed for analytics (e.g., aggregation edges and hypernodes), GRAD facilitates the explicit representation of concepts such as hierarchies and the analysis of the content of nodes. Thus proving to be a better fit for designing intra-class cubes. We use these characteristics to support dimension hierarchies and build additional OLAP cubes on graphs. We propose our novel technique for building OLAP cubes on graphs. Thereby equipping decision-makers with the capability of performing effective multi-level/multi-perspective analysis of their graph data and examining new business facts.
Our main contributions in this chapter are summarized as follows:

- We define the multidimensional concepts for graph data, and propose novel techniques for extracting the candidate multidimensional concepts and building graph cubes from property graphs.
- We present an extension of the property graph model, tailored for mul-

tidimensional analysis, and examine the additional candidate graph cubes brought by this extension. We further extend our work to support dimension hierarchies within graphs.

- We suggest a graph data warehousing architecture and provide an effective prototypical implementation of our techniques for building OLAP cubes.

The remaining of this chapter is structured as follows: Section 2 presents our running example. In Section 3, we formally define the multidimensional structures on graphs. Section 4 presents our technique for extracting potential multidimensional spaces and building graph cubes on property graphs. In Section 5 we propose a technique for building OLAP cubes on a novel graph database model, and extend our approach to support dimension hierarchies in graphs. Section 6 presents the architecture and implementation of our proof-of-concept graph warehousing framework. Finally, Section 7 sketches future work and concludes the chapter.

2 Running Example

We illustrate the analysis opportunities brought by graphs using a movie graph. The original dataset was published by the GroupLens research group.[1] The resulting graph contains movies with attributes, such as the year of release, titles, ratings, and scores from different communities, etc. Each movie is linked to its actors with an edge that contains the rank of the actor in the movie. We further enrich the dataset with information about actors' birth date and nationality, and movies country from the Movie Database website.[2] Figure 4.1-(a) shows a subgraph of the movie graph. We start with a simple and flat multidimensional schema shown in Figure 4.1-(b). We introduce in Section 5 a more complete schema supporting hierarchies and enabling more advanced analysis.

3 Multidimensional Concepts on Graphs

In this section, we formally define the multidimensional structures in the context of heterogeneous attributed graphs. We start with dimension levels.

Definition 4.1. [***Dimension Level***] *A level λ is defined by a pair $\langle name, \mathcal{P}\rangle$, where name is the name of the level, and $\mathcal{P}$ is the aggregation pattern. $\mathcal{P} = (V_P, E_P, \alpha, \beta)$, with V_P, E_P being the constraints on the level's topology, and α, β the constraints on*

[1] http://grouplens.org/datasets/movielens

[2] https://www.themoviedb.org/

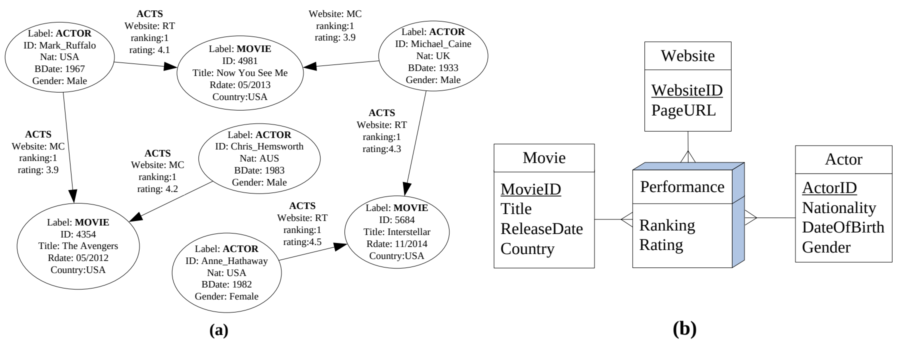

ACTS Website: RT ranking:1 rating: 4.1
Label: ACTOR ID: Mark_Ruffalo Nat: USA BDate: 1967 Gender: Male
Label: MOVIE ID: 4981 Title: Now You See Me Rdate: 05/2013 Country:USA
Website: MC ranking:1 rating: 3.9
Label: ACTOR ID: Michael_Caine Nat: UK BDate: 1933 Gender: Male
ACTS Website: MC ranking:1 rating: 3.9
ACTS Website: MC ranking:1 rating: 4.2
Label: ACTOR ID: Chris_Hemsworth Nat: AUS BDate: 1983 Gender: Male
ACTS Website: RT ranking:1 rating:4.3
Label: MOVIE ID: 4354 Title: The Avengers Rdate: 05/2012 Country:USA
ACTS Website: RT ranking:1 rating:4.5
Label: ACTOR ID: Anne_Hathaway Nat: USA BDate: 1982 Gender: Female
Label: MOVIE ID: 5684 Title: Interstellar Rdate: 11/2014 Country:USA
(a)
Website
WebsiteID
PageURL
Movie
MovieID
Title
ReleaseDate
Country
Performance
Ranking
Rating
Actor
ActorID
Nationality
DateOfBirth
Gender
(b)

Fig. 4.1: A Sample Movie Graph and Performance MD Schema

its labels and attributes respectively. $\mathcal{P}$ is used to identify all graph elements that belong to the dimension's level and that should be merged after a roll-up.

Dimensions provide possible perspectives for the analysis of the graph topology and content. In this chapter, we focus on two types of dimensions that could be derived from graph attributes: (1) Node-based dimensions, which are represented by the attributes of the nodes, and (2) Edge-based dimensions, which are represented by the attribute of the edges. We define a dimension as follows:

Definition 4.2. [***Dimension***] *A dimension is defined as $D = \langle name, \Lambda, \mathcal{R} \rangle$, where $\Lambda = \{\lambda_1, ..., \lambda_n\}$ is the set of the dimension levels. $\mathcal{R}$ is a partial order on the elements of Λ and describes a directed acyclic graph defining the hierarchy and the aggregation direction between the dimension's levels. The base level λ_1 and highest level λ_n (often referred to as All) are located at the ends of the partial order.*

In the multidimensional model, a measure is the basic unit of data that is placed in the multidimensional space and examined through the dimensions.

Definition 4.3. [***Measures***] *A measure m is identified by $\langle name, \phi \rangle$. The measure is computed by applying the aggregation function ϕ over a property graph $\mathcal{G}$ or some of its elements. In graphs, ϕ could be a traditional function such as the average of a set of attribute values, or a graph-specific function such as the degree of a node.*

Multiple classifications for graph measures were proposed in the literature, such as the classification by the aggregation type (i.e., distributive, algebraic, and holistic) [28]. Here we propose a new classification of graph measures, based on the type and the computation algorithm.

- **Content-Based Measures:** They are extracted from the attributes of graph elements. These measures are similar to traditional measures and do not capture the graph topology. For example, the average rating of a movie and the average rank of an actor are content-based measures.

- **Graph-Specific Measures:** They capture the topological properties of graphs and are obtained by applying graph algorithms. They could be classified according to the type of the output as either (1) *numerical*, where the output is a numerical value such as the value of the PageRank, or (2) *topological*, where the measure is represented using graph structures such as the path between a pair of nodes. The second possible classification makes the distinction between (1) *local* measures, which are computed separately for graph nodes or edges (e.g., the centrality of an actor), and (2) *global* measures which are computed for the whole graph (e.g., the diameter or number of cycles of the graph).

- **The Graph as a Measure:** As discussed by Chen et al. in [28], the graph itself could be considered as a measure examined from different perspectives and at different aggregation levels.

The cube metaphor is widely accepted as the underlying logical construct for conventional multidimensional models. Here we define the concept of the cube using the notion of aggregate graphs defined as follows.

Definition 4.4. [***Aggregate Graph***] *An aggregate graph $\mathcal{G}'$ of an initial graph $\mathcal{G}$ is a graph obtained by condensing a subset of the nodes and edges of $\mathcal{G}$. Hence, each node corresponds to a set of nodes in $\mathcal{G}$, and each edge is the result of the fusion of edges between pairs of aggregated nodes.*

Definition 4.5. [***Graph Cube***] *A graph cube corresponds to a set of aggregate graphs obtained by restructuring the initial graph $\mathcal{G}$ in all possible aggregations. Each cuboid is therefore represented as an aggregate graph of $\mathcal{G}$. If an aggregation is performed from λ_i to λ_{i+1}, all graph elements that satisfy the aggregation pattern $\mathcal{P}_i$ are aggregated in the same node. The edges are constructed afterward to link the pairs of nodes. Measures are then recomputed and placed on the aggregate graph.*

In the next sections, we show how these formal definitions map to the specific graph structures of each model and illustrate them with examples applied to the movie graph. We discuss how to select a valid subset of attributes as the candidate dimensions or measures, and build the different graph cubes.

4 Building OLAP Cubes on Property Graphs

Many current graph databases represent graphs using the *property graphs* model [109]. We show in this section how we can use property graphs as the first foundation for building OLAP cubes. However, since property graphs describe basic graph structures (which are simple and oriented for storage and operational workloads), their analysis capabilities are limited. For advanced multidimensional modeling and analysis, richer graph structures are needed as we show later in Section 5.

Property graphs describe a directed, labeled and attributed multi-graph. We use the same formal definition of property graphs presented in Definition 5.1 of Chapter 3.

Definition 4.6. [***Property Graph***] *Given a finite set of labels $\mathcal{L}$, a set of property keys $\mathcal{K}$ and a set of values $\mathcal{N}$, a property graph is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H})$ where:*

- *$\mathcal{V}$ is a finite set of nodes;*
- *$\mathcal{E}$ is a finite set of edges, such that $\mathcal{E} \cap \mathcal{V} = \emptyset$;*
- *$\rho : \mathcal{E} \to \mathcal{V} \times \mathcal{V}$ is the function that maps each edge to its pair of nodes, with $\rho(e) = (u, v)$ denoting a directed edge from u to v;*
- *$\mathcal{F} : \mathcal{V} \cup \mathcal{E} \to \mathcal{L}$ is the function that assigns to each node (resp. edge) a label from the set of labels $\mathcal{L}$;*
- *$\mathcal{H} : (\mathcal{V} \cup \mathcal{E}) \times \mathcal{K} \to \mathcal{N}$, is a partial function assigning to each node (resp. edge) the value of its attribute, such that, for a node (resp. edge) $v \in \mathcal{V}$, the function $\mathcal{H}(v, k_i)$ returns the value $x \in \mathcal{N}$ of its $i - th$ attribute identified by the key $k_i \in \mathcal{K}$.*

In addition to the initial definition, this chapter we represent a node $v_i \in \mathcal{V}$ as $v_i = (l_i, A_{v_i})$, where $l_i \in \mathcal{L}$ is the label and $A_{v_i} = \{(k_1, n_1), ..., (k_j, n_j) \mid k \in \mathcal{K}, n \in \mathcal{N}\}$ is the set of key-value pairs representing the attributes of v_i. Similarly, an edge $e_j \in \mathcal{E}$ is represented as $e_j = (v_s, v_e, l_j, A_{e_j})$, where $\rho(e) = (v_s, v_e)$ such that v_s and v_e are the start and end nodes respectively, $l_j \in \mathcal{L}_e$ is its label and A_{e_j} is the set of key-value pairs representing the attributes of e_j. Each node (resp. edge) on the graph has exactly one label. A **class** (denoted Σ_i) describes a set of graph nodes that share the same label.

Given a property graph $\mathcal{G}$ and a pair of nodes from two connected but distinct classes of nodes, we explore the candidate dimensions, measures, and cubes that could be built by exploring the graph of these two classes. We denote dimensions that span across two linked classes as inter-class dimensions, defined as follows.

Definition 4.7. [***Inter-class dimensions***] *Let $\mathcal{G}$ be a property graph, and let $v_s \in \Sigma_s$ and $v_e \in \Sigma_e$ be a pair of nodes from two distinct classes. Let $e_i = (v_s, v_e, l_i, A_{e_i})$*

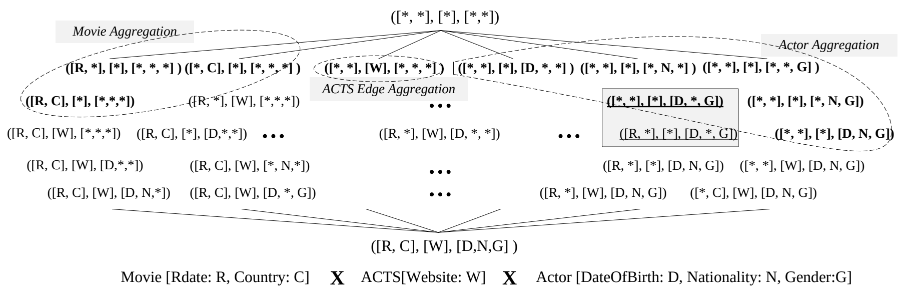


Fig. 4.2: Graph Lattice of the Movie Graph

be an edge that relates v_s and v_e. The candidate node-based dimensions are a subset of the attributes of the nodes v_s and v_e (A_{v_s} and A_{v_e}). The candidate edge-based dimensions are a subset of the attributes of the edge e_i (A_{e_i}).

Example 4.1 (Rating and Ranking of Actors per Website and Movie)
Using the running example of movies, we consider two classes which are $MOVIE$ and $ACTOR$. The movie dimensions are derived from the attributes whose keys are $\{ReleaseDate, Country\}$, and the actor dimensions are derived from the set $\{Nationality, DateOfBirth, Gender\}$. For example, following the notation of Section 3, $D_{Gender} = \langle Gender, \Lambda, \mathcal{R}\rangle$, with the levels being the base level $Gender$ and ALL. Therefore, at the base level actor nodes are grouped into two groups (i.e., nodes for male actors, and nodes for female actors), then grouped in one node regardless of the gender (i.e., at the ALL level represented often by $*$). The edge-based dimension is represented by the attribute $Website$ of the edge $ACTS$ relating actors and movies.

The graph lattice enumerates all possible OLAP aggregations of the graph and is obtained by aggregating over all the inter-class dimensions. Given the graph of Figure 4.1, and considering the dimensions of the previous example, Figure 4.2 shows the graph lattice of the performance cube. Each node of the graph lattice represents an aggregate graph, that is, a cuboid of the graph cube. We distinguish three special kinds of aggregation on this graph (highlighted in Figure 4.2), which are *Movie*-only aggregations (i.e., only movie nodes are kept not fully aggregated to the *All* level), *ACTS*-only aggregation, and *Actor*-only aggregations.

Definition 4.8. [***Inter-class measures***] *Given a property graph $\mathcal{G}$ and a set of edges $E \subseteq \mathcal{E}$ relating nodes of the classes Σ_i and Σ_j, a content-based measure m_c is*

computed by applying an aggregation function ϕ on an attribute of the edges $e_i \in E$
.

The graph-specific measures are obtained by applying graph algorithms on $\mathcal{G}$. Also, the aggregated graph itself could be considered as a measure.

To analyze the properties of the relationships between the graph entities, inter-class modeling focus on the potential dimensions and measures existing within the entity edges. We cannot assume that all attributes of the edges are dimensions. The distinction between attributes that are dimensions and attributes that are measures is not straightforward, and thus requires a modeling effort from the designer to distinguish them.

Example 4.2 (Multidimensional Aggregation of the Movie Graph)
As shown by the multidimensional schema of Figure 4.1-(b), the attribute *Website* of the edge labeled *ACTS* could indeed be a dimension. However, the attributes *ranking* and *rating* are rather considered as measures in the current analysis scenario.
We apply these dimensions and measures on the property graph of Figure 4.1-(a), and follow the graph lattice of Figure 4.2 to study the graph cube reflecting the ranking and rating of actors in the movie graph. Figure 4.3-(a) shows the aggregate graph (i.e., graph cuboid) where movies are grouped by release date, and actors are grouped by birth date and gender. A corresponding OLAP cube is shown in Figure 4.3-(b). The measures are *AverageRanking* and *AverageRating* of actors, which can be examined through the three dimensions left (i.e., *ReleaseDate*, *DateOfBirth*, and *Gender*). We follow the graph aggregation as depicted by Figure 4.3-(e) to get the graph (Figure 4.3-(c)) and the cuboid (Figure 4.3-(d)) at the next aggregation level. On the lattice of Figure 4.2, this aggregation corresponds to the two nodes underlined and put in a rectangle. Note here that for graph-specific measures (e.g., closeness centrality of actors), the measures for the upper-level could not be computed directly from the cube at a lower level, as the computation function needs to traverse the aggregated graph itself to compute the new value of the graph-specific measure.

5 Building OLAP Cubes on GRAD

Many graph models were proposed in the literature to abstract different types of graphs and fit their particular analysis workloads [10]. In [50], we proposed GRAD, an analysis-oriented graph database model that extends property graphs with advanced graph structures, integrity constraints, and a

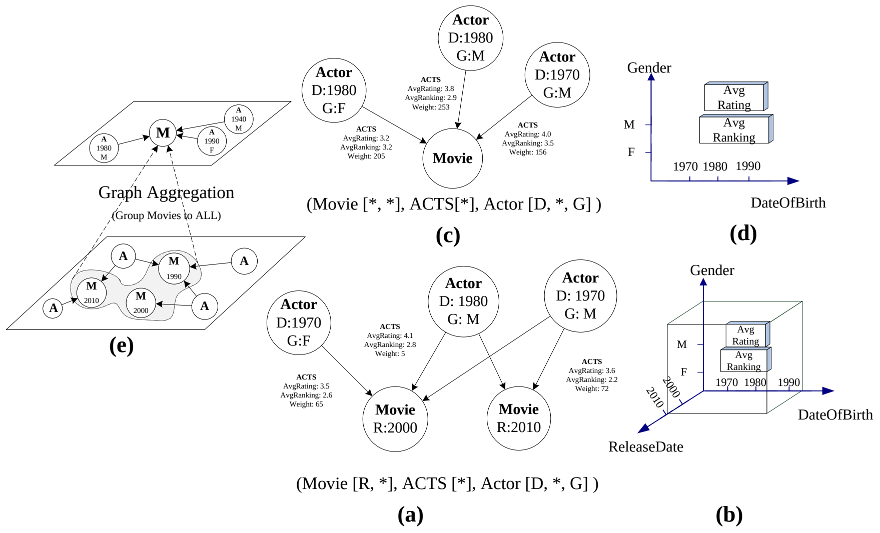

Actor D:1980 G:M
Actor D:1980 G:F
Actor D:1970 G:M
ACTS AvgRating: 3.8 AvgRanking: 2.9 Weight: 253
ACTS AvgRating: 3.2 AvgRanking: 3.2 Weight: 205
ACTS AvgRating: 4.0 AvgRanking: 3.5 Weight: 156
Movie
(Movie [*, *], ACTS[*], Actor [D, *, G])
(c)
Gender
M
F
Avg Rating
Avg Ranking
1970 1980 1990
DateOfBirth
(d)
A 1940 M
A 1980 M
A 1990 F
M
Graph Aggregation
(Group Movies to ALL)
A
M 1990
M 2010
M 2000
(e)
Actor D:1970 G:F
Actor D: 1980 G: M
Actor D: 1970 G: M
ACTS AvgRating: 4.1 AvgRanking: 2.8 Weight: 5
ACTS AvgRating: 3.5 AvgRanking: 2.6 Weight: 65
ACTS AvgRating: 3.6 AvgRanking: 2.2 Weight: 72
Movie R:2000
Movie R:2010
(Movie [R, *], ACTS [*], Actor [D, *, G])
(a)
Gender
ReleaseDate
2000
2010
DateOfBirth
(b)

Fig. 4.3: OLAP Aggregation of the Movie Graph and Computation of the OLAP Cubes

graph algebra. We use GRAD as the foundation for the OLAP cubes extraction techniques we present in this section.

As we discussed in the previous section, property graphs support OLAP analysis of inter-classes facts. However, they fall short from supporting OLAP analysis of the internal information stored within each node, or class of nodes. Therefore, we focus in this section on the additional cubes and analysis capabilities brought by GRAD. Note however that since GRAD extends property graphs, the candidate multidimensional spaces and cubes discussed in the previous section could similarly be built using GRAD.

5.1 OLAP Cubes on GRAD

In GRAD, we consider heterogeneous, attributed, and labeled graphs. Complex attributes are supported on the nodes and rich semantics is explicitly expressed on the edges. The analysis process is centered around special analytical structures, namely *hypernodes* and *classes*. Hypernodes represent real-world entities and are grouped within classes. Each analytics hypernode is an induced subgraph grouping an entity node, all its attribute and literal nodes, and all the edges between them. The core of a hypernode is the entity node that contains the label and the identifier attributes of the real-world entity. Attribute nodes are attached to the entity node and denote the non-identifier, and potentially multi-valued attributes of each entity (e.g., budget, revenue). Literal nodes record the effective value of its corresponding attribute node. Rich semantics are embedded on the graph edges such as multiplicities, hierarchical, and composition relationships. A complete definition of GRAD is

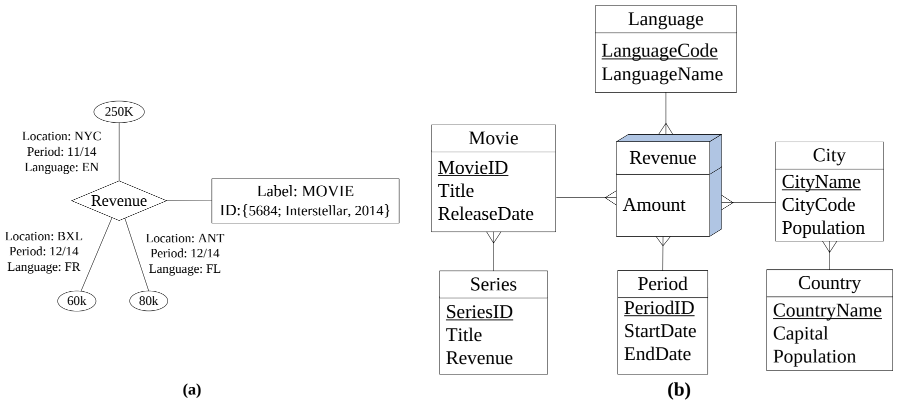


Fig. 4.4: Movie GRAD Graph and Revenue MD Schema

provided in Chapter 3.

Formally, given a finite set of labels $\mathcal{L}$, a set of property keys $\mathcal{K}$ and a set of values $\mathcal{N}$, a graph in GRAD denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H})$, and a class of entity nodes is denoted as Σ_i. $\mathcal{V} = V_e \cup V_a \cup V_l$, with V_e being the set of entity nodes, V_a the set of attribute nodes, and V_l the set of literal nodes. The set of edges is $\mathcal{E} = E_e \cup E_a \cup E_l$, with V_e being the set of entity edges, E_a the set of attribute edges, and E_l the set of literal edges.

Figure 4.4-(a) illustrates a part of the movie graph modeled with GRAD. In this example, *Movie* is an entity node, while *Revenue* is an attribute node attached to *Movie*. The revenue has different values depending on a set of factors (location, time, language, etc.), and each value is stored separately in a literal node.

In the previous section, we used property graphs to study the candidate multidimensional cubes between classes of nodes. As GRAD is an extension of property graphs, the same modeling abd analysis of intra-class cubes could be achieved using GRAD. Therefore, in this section we explore the additional candidate dimensions, measures, and cubes that could be extracted from a single class Σ_i using the extension proposed by GRAD.

Definition 4.9. [***Intra-Class Dimension***] *Given a GRAD graph $\mathcal{G}$, a class of entity nodes Σ_i, and an entity node $u \in \Sigma_i$ with ID_u being the set of identifier attributes of u. Then we can extract distinct sets of candidate dimensions. Each set of dimensions is the union between the attributes of the entity node and the attributes of the literal edge of a given attribute node. For a given attribute node $v_i \in V_a$ linked to the entity node u, where A_i is the attributes of the literal edge $e \in E_l$ corresponding to v_i, the set of dimension us $D_{v_i} = \{ID_u\} \cup A_i$.*

Definition 4.10. [***Intra-class measures***] *Given a GRAD graph $\mathcal{G}$, an intra-class measure is defined by $\langle name, \phi \rangle$ and is explored within each hypernode. The label*

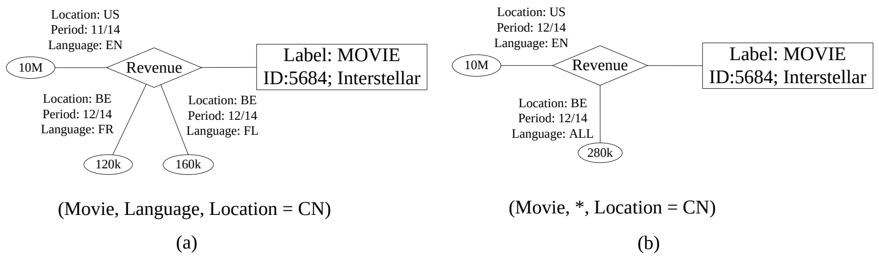


Fig. 4.5: Aggregation of Revenue by Language

of the attribute node is the name of the measure (name $\in L_a$*). The actual values of these measures are embedded as the attributes of the literal nodes.*

Example 5.1 (Analysis of the Revenue of a Movie)
Given the example of Figure 4.4, suppose an analyst needs to examine the revenue of movies following the multidimensional schema of Figure 4.4-(b). Revenue is therefore considered as the name of the measure, which is the same as the label of the attribute node *Revenue*. The aggregation function is *SUM*. The values of the measures are stored within the literal nodes linked to the *Revenue* attribute node and the function computing the measure is the same as the one used to retrieve the value from the literal node. The dimensions for the revenue measure are named *Movie*, *Location*, *Period*,, and *Language*. Given these dimensions, we can aggregate the graph to examine the value of revenue by navigating through the dimension hierarchy of the *Location* dimension from *City* to *Country* as shown in Figure 4.5-(a), or by rolling up to the level *ALL* of the language dimension as in Figure 4.5-(b). Concretely, at the graph level, this operation will incur merging the corresponding literal storing the measure values.

We distinguish here two types of graph aggregations: (1) Intra-hypernode aggregation, where literal nodes and edges of the same attribute node are merged, thus the dimensions is an attribute of the literal edges (e.g., revenue of a given movie by language), (2) Inter-hypernode aggregation, where entity nodes could be merged (e.g., revenue of all movies per given a city, period and language).

5.2 Dimension Hierarchies on GRAD

In this subsection, we consider extending the OLAP analysis to support hierarchies within inter-class and intra-class dimensions.

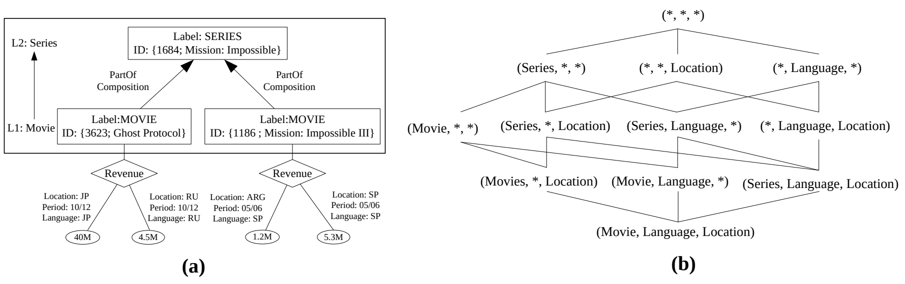


Fig. 4.6: Dimension Hierarchy within Movie's Class

- Dimension hierarchy for intra-class dimensions: Within each dimension (i.e., attribute location of revenue), we might have an inner hierarchy (e.g., City, Region, and Country). Therefore, we can extend the lattice with these new possible aggregations as shown in Figure 4.5-(a).

- Dimension hierarchy for inter-class dimensions: Explored between distinct classes of nodes. Within GRAD, specific types of edges such as composition and aggregation could be explicitly defined. Therefore, classes of nodes related by these specific relationships belong to the same dimension with the hierarchy following the child-parent direction of these relationships. Figure 4.6-(a) shows the hierarchy of the movie dimension that is now composed of *Movie* and *Series* levels. The updated lattice is shown in Figure 4.6-(b).

6 Framework Architecture and Implementation

In this section, we present our prototypical implementation of the OLAP cubes extraction approach using Neo4j. The framework architecture is depicted in Figure 4.7. The major components of our implementation are described as follows:

1. Graph ETL: The graph is extracted from external data sources that might have various formats (e.g., XML as for DBLP, or text files for MovieLens, etc.). For the running example, we have developed two modules for extracting and matching data from CSV files of MovieLens with data about actors from The Movie Database. The data is then formatted following GRAD and property graph structures before being loaded as the base graph on Neo4j.

2. Graph storage and materialization: The graph data is stored using multiple Neo4j graph database instances. We use two particular databases,

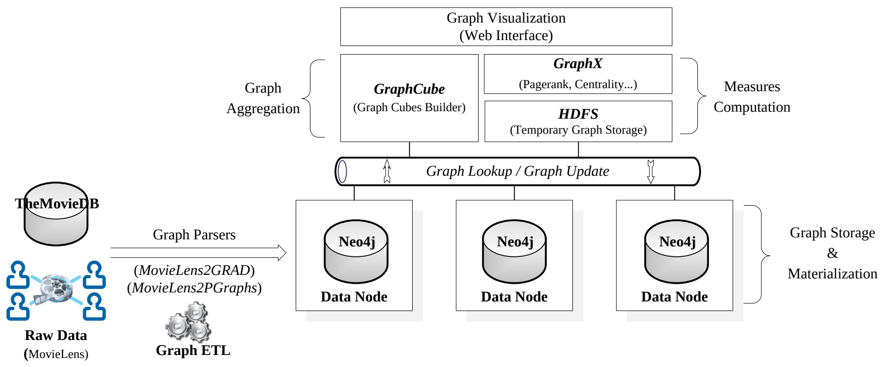


Fig. 4.7: Distributed OLAP Cubes Computation

one to store the graph at the base level and the other to keep the lattice. The other instances store the aggregate graphs. However, we needed a database-per-aggregate graph because Neo4j does not support materialized views on graphs, and could not separate between subgraphs of the same database.

3. Graph lookup and update: This component acts as a middleware between the storage and processing layers. It loads the graph, at a given aggregation level, from a Neo4j database into HDFS to prepare it for distributed processing or aggregation. Once the processing is done, this layer stores the graph back into a new Neo4j instance if the graph was aggregated, or updates the original database if only some attributes were updated.

4. Graph Aggregation and Measures Computation: Given a graph lattice, the *GraphCube* module performs the graph aggregation to generate potential graph cuboids as discussed through this chapter. To efficiently compute the graph-specific measures (e.g., PageRank or closeness centrality), we use the GraphX library. GraphX performs the iterative graph algorithms in-memory and thus outperforms the other distributed graph libraries on large scale graphs. Once the required graph measures are computed, the result is persisted in the corresponding Neo4j instance using the previous layer.

7 Conclusion

In this chapter, we proposed our contribution to graph warehousing by designing novel techniques for building OLAP cubes on graphs. We applied our approach to both property graphs and a more advanced graph database model tailored for multidimensional modeling. We discussed techniques for OLAP aggregation of the graph and tackled the case of dimension hierarchies in graphs. Besides, we provided an overview of the architecture and implementation of our graph warehousing framework.

Graph data warehousing is an emerging research field that brings various challenges similar to traditional data warehousing (e.g. high dimensionality and cubes materialization). However, the structural properties and unstructured nature of graphs call for the development of novel modeling and processing paradigms. Our immediate future work is to enable multidimensional concepts discovery on graphs within our framework. Yet, many remaining research directions are worth investigating to build industry-grade graph warehousing systems. Among these directions, we cite OLAP analysis of dynamic graphs and the definition of a proper OLAP algebra and query language for graphs.

Chapter 5

TopoGraph: An End-To-End Framework to Build and Analyze Graph Cubes

The paper has been published in the *Information Systems Frontiers Journal.* The layout and content of the paper have been revised.
DOI: https://doi.org/10.1007/s10796-020-10000-z

Abstract

Graphs are a fundamental structure that provides an intuitive abstraction for modeling and analyzing complex and highly interconnected data. Given the potential complexity of such data, some approaches proposed extending decision-support systems with multidimensional analysis capabilities over graphs. In this chapter, we introduce TopoGraph, an end-to-end framework for building and analyzing graph cubes. TopoGraph extends the existing graph cube models by defining new types of dimensions and measures and organizing them within a multidimensional space that guarantees multidimensional integrity constraints. This results in defining three new types of graph cubes: property graph cubes, topological graph cubes, and graph-structured cubes. Afterward, we define the algebraic OLAP operations for such novel cubes. We implement and experimentally validate TopoGraph with different types of

real-world datasets.

1 Introduction

Many approaches were proposed to address the graph data warehousing challenge [48, 103]. These efforts laid the foundation for multidimensional modeling and analysis of graphs both at the modeling and physical levels. In this chapter, we extend the state of the art on graph warehousing by introducing new types of graph cubes that leverage both the content and the topology of the graphs and expose topological and graph-structured insights. Most state-of-the-art papers consider the aggregate graph as the only measure to be examined. A notable difference of our work compared to these papers is that we (1) capture new types of measures at a finer granularity level, (2) represent them individually with numerical values or graphs, and (3) position them within new types of graph cubes. Therefore, these cubes embed new types of measures and dimensions not captured by previous work. We discuss the required multidimensional integrity constraints on graphs, completely overlooked by current approaches, and show that our cubes guarantee them. To the best of our knowledge, our framework is the first to define and guarantee the multidimensional integrity constraints on graphs. We further discuss the correspondence between graph cubes and relational OLAP cubes and identify the few specific cases where a graph cube could be loaded into a ROLAP cube. We show that the integration of graph cubes with the existing ROLAP systems is not a trivial task and motivate the need for graph-specific warehousing systems.

The research questions we address in this chapter are: *given a graph, what kind of new graph cubes could be extracted from it? When could a graph cube be mapped and loaded to a ROLAP cube? And how could such novel graph cubes be analyzed from a multidimensional perspective?* As a result, we present **TopoGraph**, a graph data warehousing framework that extends current graph warehousing models with new types of cubes and queries combining graph-oriented and OLAP querying. TopoGraph goes beyond traditional OLAP cubes, which process value-based grouping of tables, by considering the topological properties of the graph elements. And it goes beyond current graph warehousing models by proposing new types of graph cubes. These cubes embed a rich repertoire of measures that could be represented with numerical values, with entire graphs, or as a combination of them. Moreover, we discuss the correspondence between the graph cubes proposed in this chapter and traditional OLAP cubes and motivate the need for native graph warehousing systems. Given the proposed cubes, TopoGraph aims at providing answers to complex questions, asked in a business context, that require the analysis of both the content and the topology of the graph. Relevantly, TopoGraph is our proposal to overcome the current shortcomings of graph warehousing approaches resulting from our experience in real-life enterprise settings. We illustrate in the following example three typical questions to which TopoGraph is designed to answer.

Example 1.1 (Social Network)
Consider a social network such as Twitter where a set of users are following each other, post and retweet a set of tweets as illustrated in Figure 5.1. We distinguish three types of queries that could be used for analyzing graph properties from a multidimensional perspective and illustrate them on a social network through the following examples.

Content Query Content query target content-based measures and are computed by applying aggregation functions such as count and average. They are used to answering queries such as (1) counting the total number of favorites a tweet received from a certain user location, (2) the average number of followers a community of users has, or (3) the total number of tweets on a given date by users in a certain language. Existing OLAP frameworks are designed to handle this type of query.

Topological Query These queries focus on the topological properties of graph elements and are computed by applying graph algorithms that output numerical values, such as node degree, graph diameter, and PageRank. These queries take as input a graph and return a numerical value. They could be used to answer questions such as finding the most influential (i.e., having the highest PageRank) users from a given community (i.e., after computing a community detection algorithm), who tweeted in a certain language. This kind of query is specific for graph cubes, and they cannot be naturally supported by traditional OLAP frameworks. Graph analytics frameworks are the natural choice to tackle this type of queries, but current graph warehousing methods do not provide foundations on how to combine them with OLAP.

Graph-Structured Query Graph structured queries use graph patterns to match and retrieve complex graph information. Both the input and the output of these queries are graphs. These queries are computed by applying graph algorithms that output graphs, such as frequent pattern mining and minimum spanning tree to capture complex grouping of graph elements. They could be used to answer questions such as (1) finding the most frequent communication pattern in a network of users from a given location, or (2) retrieving the shortest path between a pair of tweets

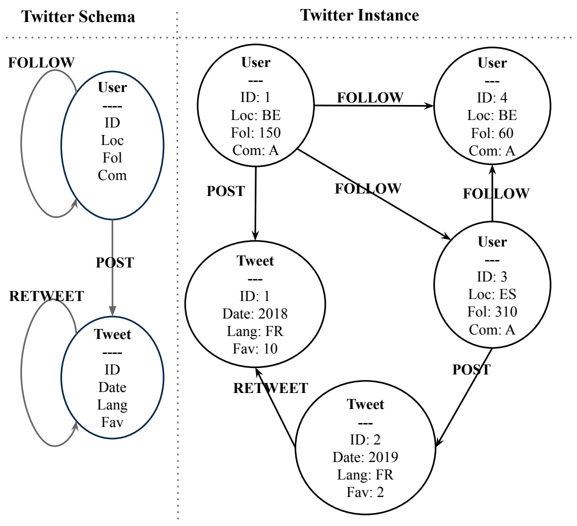


Fig. 5.1: Schema and Instance of a Social Network Property Graph

during a certain period. Similar to the previous case, the current graph warehousing approaches do not support these kinds of queries.

Our main contributions are summarized as follows:

- We propose TopoGraph, a novel graph warehousing model aware of the topological properties of graphs, to support decision-making on graphs. Thus, we formally define three novel categories of graph cubes: property graph cubes, topological graph cubes, and graph-structured cubes. These cubes capture both content and topological properties of heterogeneous graphs. They define new types of measures and dimensions specific for graphs and place them within the multidimensional space while preserving the multidimensional integrity constraints on graphs.
- We discuss the few cases where the information captured by the proposed graph cubes could be loaded into corresponding OLAP cubes. We show that there is a big gap between graph cubes and relational OLAP cubes, and motivate the need for dedicated graph warehousing systems.
- We define the algebraic OLAP operations for the new graph cubes, and illustrate their application for querying the topological information embedded in the graphs. These operators enable complex graph querying

and OLAP analysis of the topology and content of graph cubes from different perspectives and through different aggregation levels.

- We implement the novel graph cube computation and aggregation approach proposed by TopoGraph, and experimentally validate its efficiency with different types of real-world datasets. We further describe the architecture and the API of a social network analysis framework built with TopoGraph.

We organize the rest of the chapter as follows. In Section 2 we formally define the property graph cube and its related multidimensional concepts. In Section 3, we define the concept of topological graph cubes and detail the particular process of deriving OLAP Cubes containing topological measures. Section 4 introduces graph structured cubes. We define OLAP operations on graph cubes in Section 5. In Section 6, we describe the architecture of a graph warehousing system using TopoGraph and evaluate its performance on Neo4j using multiple datasets. Finally, Section 7 discusses open challenges and concludes the chapter.

2 Graph Cubes on Property Graphs

In this section, we discuss the graph data model, and we introduce the multidimensional model for graphs, used for building and analyzing the graph cubes.

2.1 Property Graphs

Property graphs are a widely accepted model for graph representation and are implemented by established graph databases such as Neo4j. They describe a directed, labeled, and attributed multi-graph [109]. Each real-world entity is represented by a node. Relationships between entities are represented using edges. We reuse again in this chapter the same formal definition of property graphs presented in Definition 5.1 of Chapter 3.

Definition 5.1. [***Property Graph***] *Given a finite set of labels $\mathcal{L}$, a set of property keys $\mathcal{K}$ and a set of values $\mathcal{N}$, a property graph is represented as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H})$ where:*

- *$\mathcal{V}$ is a finite set of nodes;*
- *$\mathcal{E}$ is a finite set of edges, such that $\mathcal{E} \cap \mathcal{V} = \emptyset$;*
- *$\rho : \mathcal{E} \to \mathcal{V} \times \mathcal{V}$ is the function that maps each edge to its pair of nodes, with $\rho(e) = (u, v)$ denoting a directed edge from u to v;*

- *$\mathcal{F} : \mathcal{V} \cup \mathcal{E} \rightarrow \mathcal{L}$ is the function that assigns to each node (resp. edge) a label from the set of labels $\mathcal{L}$;*
- *$\mathcal{H} : (\mathcal{V} \cup \mathcal{E}) \times \mathcal{K} \rightarrow \mathcal{N}$, is a partial function assigning to each node (resp. edge) the value of its attribute, such that, for a node (resp. edge) $v \in \mathcal{V}$, the function $\mathcal{H}(v, k_i)$ returns the value $x \in \mathcal{N}$ of its $i-th$ attribute identified by the key $k_i \in \mathcal{K}$.*

As in Chapter 4, in addition to the initial definition, in this chapter we represent a node $v_i \in \mathcal{V}$ as $v_i = (l_i, A_{v_i})$, where $l_i \in \mathcal{L}$ is the label and $A_{v_i} = \{(k_1, n_1), ..., (k_j, n_j) \mid k \in \mathcal{K}, n \in \mathcal{N}\}$ is the set of key-value pairs representing the attributes of v_i. Similarly, an edge $e_j \in \mathcal{E}$ is represented as $e_j = (v_s, v_e, l_j, A_{e_j})$, where $\rho(e) = (v_s, v_e)$ such that v_s and v_e are the start and end nodes respectively, $l_j \in \mathcal{L}_e$ is its label and A_{e_j} is the set of key-value pairs representing the attributes of e_j. Each node (resp. edge) on the graph has exactly one label. $\mathcal{A} = \{A_1, .., A_p\}$ is the set of attributes of the graph elements so that for each node $v \in \mathcal{V}$ (resp. edge $e \in \mathcal{E}$) the set of its j attributes is $\mathcal{A}(v) = \{A_1(v), A_2(v), ..., A_j(v)\}$, with $j \leq p$.
Figure 5.1 shows an example of a social network property graph. The graph represents a set of Twitter user nodes and their tweets. In addition to its attributes (e.g., Location or Language), each graph element has a label, which is the distinctive attribute that defines the type of the graph element (e.g., User, Tweet, FOLLOW, etc.).

2.2 Property Graph Cubes

Using the property graphs model introduced in the previous section, we define the multidimensional concepts in the context of graph data. These definitions are based on and extend the graph cube definitions presented in the previous works [143]. We formalize the concept of dimension, measure, and graph cube. The multidimensional concepts introduced in the section focus on capturing the content-based information of the graph and are therefore similar to the traditional multidimensional concepts used by relational frameworks.

Definition 5.2. [***Dimension***] *A dimension provides the possible perspectives for the multidimensional analysis of the graph topology and content. It is defined as $D_i \in \mathcal{D}$, where $\mathcal{D} \subseteq \mathcal{A}$ is a subset of the attributes of the graph elements. Dimension attributes have to belong to a discrete domain. Given a graph element $u \in \mathcal{V} \cup \mathcal{E}$, the set of its n dimensions is $\mathcal{D}(u) = \{D_1(u), ..., D_n(u)\}$.*

Definition 5.3. [***Dimension Hierarchy***] *A set of dimensional attributes might be linked by a hierarchy defined by the triple $\langle name, \Lambda_{dim}, \mathcal{R} \rangle$. $\Lambda_{dim} = \{\lambda_1, ..., \lambda_n\}$*

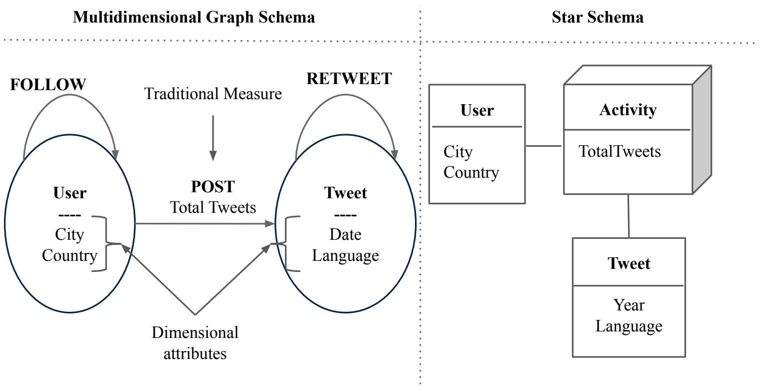


Fig. 5.2: Tweeting Activity Multidimensional Graph and Star Schema

represents the set of the hierarchical dimensional levels of a dimension $dim \in \mathcal{D}$. $\mathcal{R}$ is a partial order on the elements of Λ_{dim} and describes a directed acyclic graph defining the hierarchy between the dimension's levels. The base level and highest level $Apex$ (denoted with $*$) are located at the end of the partial order.

Definition 5.4. [***Measure***] *A measure is the basic unit of data that is placed in the multidimensional space and is examined through the dimensions. It is defined as $M_i \in \mathcal{M}$, where $\mathcal{M} \subseteq \mathcal{A}$, and each M_i is the result of applying an aggregation function ϕ (e.g., SUM, AVG etc.) on a set of attributes of the graph. Given a graph element $u \in \mathcal{V} \cup \mathcal{E}$, the set of its n measures is $\mathcal{M}(u) = \{M_1(u), ..., M_n(u)\}$.*

Multiple classifications for measures were proposed in the literature [127], such as the classification by the aggregation function. Depending on the aggregation function a measure could be (1) distributive (i.e., the function could be executed in a distributive way such as count or sum), (2) algebraic (i.e., the function is a combination of distributive ones such as average), or (3) holistic (the measure needs to be recomputed from scratch such as the median).

Definition 5.5. [***Multidimensional Graph***] *Given a property graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H})$, we represent a multidimensional graph as $\mathcal{G}_M = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H}, \mathcal{D}, \mathcal{M})$. $\mathcal{G}_M$ is a property graph annotated with multidimensional concepts. That is, dimensional semantics are added to the aggregated graph elements by selecting the attributes that are considered as dimensions, and those considered as measures such that $\mathcal{D} \cup \mathcal{M} \subseteq \mathcal{A}$ and the set of dimension and measure attributes are disjoint $\mathcal{D} \cap \mathcal{M} = \emptyset$.*

An aggregate graph $\mathcal{G}'_M$ of a multidimensional graph $\mathcal{G}_M$ is obtained by merging a subset of the nodes and/or the edges of $\mathcal{G}_M$. Only nodes (resp. edges) with the same labels can be merged together. The measures of the

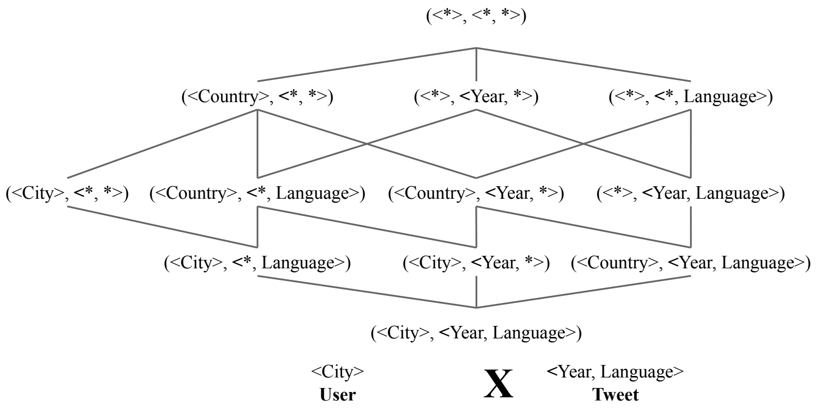


Fig. 5.3: Tweeting Activity Lattice

aggregate nodes (resp. edges) are computed using an aggregation function applied on the corresponding attributes of the nodes (resp. edges) of the initial graph. Formally, multidimensional graph aggregation is defined as follows:

Definition 5.6. [***Multidimensional Aggregate Graph***] *Given a multidimensional graph $\mathcal{G}_M = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H}, \mathcal{D}, \mathcal{M})$, an aggregate graph $\mathcal{G}'_M = (\mathcal{V}', \mathcal{E}', \rho, \mathcal{F}, \mathcal{H}, \mathcal{D}', \mathcal{M}')$ of $\mathcal{G}_M$ is obtained by aggregating $\mathcal{G}_M$ along a subset of the dimensions $\mathcal{D}' \subset \mathcal{D}$. The aggregate multidimensional graph $\mathcal{G}'$ is defined as follows:*

- *$\mathcal{V}'$ is the set of nodes, where each node $v' \in \mathcal{V}'$ is an aggregate node associated with a group of vertices $G_v \subseteq \mathcal{V}$. $G_v = \{v_i, .., v_k\}$ is obtained by selecting the nodes that share the same dimension value that is being used for aggregation;*

- *$\mathcal{E}' \subseteq \mathcal{V}' \times \mathcal{V}'$ is the set of edges, where each edge $e' = (u', v') \in \mathcal{E}'$ is an aggregate edge associated with a group of edges $G_e \subseteq \mathcal{E}$. $G_e = \{e_i, .., e_k\}$ is obtained by selecting the edges that share the same dimension value that is being used for aggregation;*

- *The dimensions of the aggregate graph are $\mathcal{D}' \subset \mathcal{D}$. $D'_i = D_i$ if the attribute D_i was not aggregated (e.g. community remains the same after grouping users by community), and $D'_i = \emptyset$, often represented with a $*$ in the literature, if the dimension D_i is removed after the aggregation (e.g., user ages are removed after grouping users by community);*

- *The measures of the aggregate graph are $\mathcal{M}' = \{M_1', M_2', ..., M_n'\}$. The value of the measure M'_i of an aggregate node (resp. edge) v' is computed by applying an aggregation function ϕ on the corresponding attribute values* *of all the nodes of G_v (corresponding to v'), formally $M'_i(v') = \phi_i(G_v)$ (e.g., computing the total number of followers per community node after grouping users by community).*

Definition 5.7. [***Property Graph Cube***] *A property graph cube is the fundamental structure supporting the multidimensional modeling and analysis of the graph data. It consists of multiple graph cuboids, each of which is a multidimensional aggregate graph built by aggregating the original multidimensional property graph using the dimensional attributes. The lattice is used to represent and organize all the possible multidimensional aggregations of the graph. Graph cuboids relate between them when a cuboid contains an attribute with a roll-up relationship, i.e., belong to the same dimension hierarchy and are directly related. Given n dimensional attributes, the graph cube contains 2^n graph cuboids that could be aggregated following the lattice structure. We distinguish two particular graph cuboids: (1) the base graph cuboid (where the multidimensional graph is at the base level), and (2) the apex graph cuboid (where the multidimensional graph is aggregated to the top-level).*

Multidimensional aggregation of a property graph is the operation of consolidating a set of graph elements into a single one located at a higher level of the lattice. Two constraints need to be enforced when building graph cubes: (1) correct aggregation of the graph cuboids, and (2) correct placement of the graph measures within the multidimensional space. To ensure a correct aggregation of cube measures along dimension hierarchies, the graph aggregation should satisfy three constraints [80]:

1. Completeness: Dimensional concepts are embedded within the graph. Therefore, every graph element should be involved in at least one dimension hierarchy. During a multidimensional aggregation, all matched graph elements should be aggregatable to a higher dimension hierarchy level. This constraint is satisfied if every graph element is associated with at least one dimension level;

2. Disjointness: Each graph element is included at most once to create an aggregate entity. This condition is satisfied if every graph element could not belong to more than one dimension level at once. Completeness and disjointness are complementary properties that could be formally defined for a multidimensional graph aggregation as follows: $\forall v \in G_v$ there exists one and only one aggregate node $v' \in \mathcal{V}'$ corresponding to the set of nodes G_v, and $\forall e(u,v) \in G_e$, there exists one and only one $e' \in \mathcal{E}'$ corresponding to G_e;

3. Compatibility: Compatibility between the aggregation algorithm and the aggregate graph elements to prevent non-meaningful operations such as computing the sum of user ages. The compatibility depends

on the application. For example, when aggregating a group of users, the designer can decide whether applying an aggregation function such as the average of the PageRank is meaningful for the application. Typically, compatibility requires additional external knowledge to know what metrics can be aggregated with what functions.

Completeness and disjointness are guaranteed by a one-to-one relationship between aggregatable and aggregate elements between consecutive cuboids. TopoGraph cubes guarantee this constraint as there exist *one and only one* node $v' \in \mathcal{V}'$ (resp. edge) in an aggregate graph corresponding to any given set of nodes G_v in the original graph. Compatibility cannot be automatically checked unless additional information is provided. On the other hand, the placement constraint is guaranteed given that the set of dimensional values generating the multidimensional space is different for each graph measure. Therefore, at most one graph element that contains a certain combination of dimension values exist in a given graph cuboid. This constraint is enforced by the statement that any pair of graph elements that have the same label and the same attributes A_i' are merged when aggregating a graph. In the same vein, the following types of graph cubes introduced in this chapter satisfy the multidimensional integrity constraint, as they follow a similar methodology for building and aggregating the graph.

Example 2.1 (Popularity Graph Cubes)
Given the Twitter property graph of Figure 5.1, we design a possible multidimensional graph schema reflecting the tweeting activity of users. Figure 5.2 depicts the multidimensional graph schema and the corresponding star schema of the OLAP cube, while Figure 5.3 depicts its lattice. The dimensions for *User* and *Tweet* nodes are $\mathcal{D} = \{City, Country, Year, Language\}$, with the hierarchical levels $\Lambda_{location} = \{City, Country\}$. The measure is computed on the *POST* edge $\mathcal{M} = \{TotalTweets\}$.

3 Topological Graph Cubes

The analysis of content-based properties of graph data (e.g., compute the average number of favorites of tweets of a given user group) is similar to the OLAP analysis of relational data in that it does not exploit the graph structure. We focus therefore in the following sections on the two graph-specific cubes introduced in Section 1: topological and graph-structured cubes.

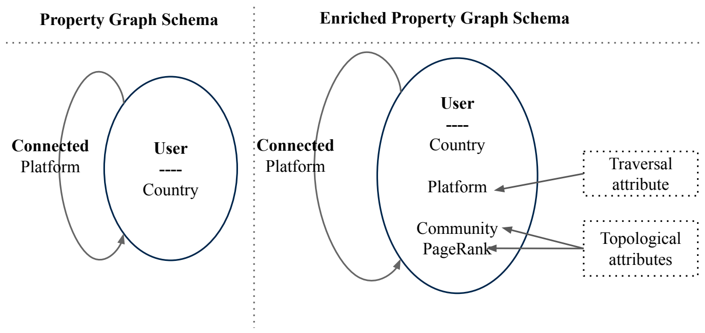


Fig. 5.4: Enriching Property Graph with Topological Attributes and Deriving Multidimensional Schema

3.1 Topological Cube Model

A rich repertoire of algorithms was developed to efficiently compute topological measures such as the centrality of nodes, or community of users. These techniques can reveal interesting properties about the graph topology and the connectivity between graph elements. Indeed, modeling data as a graph is typically done when there is an interest in exploiting such techniques. We define in this section the concept of topological graph cubes, and use them to model and analyze topological graph properties from a multidimensional perspective. As a consequence, this kind of cubes merges graph analytics and OLAP. We define first the topological concepts and discuss how to derive them from a given property graph.

Definition 5.8. [***Topological Attribute***] *Given a graph $\mathcal{G}_M = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H}, \mathcal{D}, \mathcal{M})$, a topological attributes is defined as ${A_i}^t \in \mathcal{A}$. The value of the topological attribute ${A_i}^t$ for a node $v \in \mathcal{V}$ (resp. an edge $e \in \mathcal{E}$) is given by ${A_i}^t(u) = \mathcal{T}(v, l)$ where:*

- *$\mathcal{T}$: is the function computing the topological attribute value for v (resp. e). This function relies on a graph algorithm (such as Louvain for community detection) to compute the value of the topological attribute ${A_i}^t$ for the node v;*
- *$l \in \mathcal{L}$: most graph algorithms are designed to traverse a homogeneous graph to compute the topological attributes. However, this chapter addresses the general case of heterogeneous graphs. The label l is used to guide the algorithms through a homogeneous subgraph of the input graph.*

Given a property graph, an enriched graph could be obtained by applying graph algorithms to add more topological properties to the nodes and edges before performing OLAP. For example, Figure 5.4 shows the schema of the initial property graph and an enriched version of it where two topological attributes were computed and integrated into the graph. We distinguish three different categories of attributes:

- Traditional attributes: reflect content-based properties of the graph elements such age *Country* of users.
- Topological attributes: reflect topological properties of the graph elements such as *community* and *PageRank* of users.
- Traversal attributes: contain the label of the edge traversed by the graph algorithm to compute the topological attributes (e.g. *Platform* for user nodes used for computing community and PageRank).

Definition 5.9. [***Topological Dimensions***] *Given a graph $\mathcal{G}_M = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H}, \mathcal{D}, \mathcal{M})$, the topological dimensions $D^t \in \mathcal{D}$ are a subset of the topological attributes $D^t \subseteq \mathcal{A}^t$ used for analyzing the topological graph properties from different perspectives and at different granularities.*

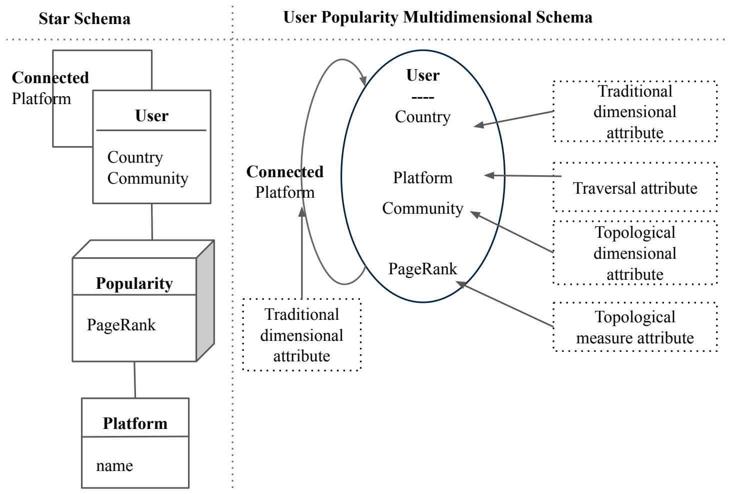
Definition 5.10. [***Topological Measures***] *Given a graph $\mathcal{G}_M = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H}, \mathcal{D}, \mathcal{M})$, the topological measures $M^t \subseteq \mathcal{M}$ are a subset of the topological attributes $M^t \subseteq \mathcal{A}^t$ analyzed in the graph cube.*

The set of topological dimensions and measures form the set of topological attributes: $D^t \cup M^t = \mathcal{A}^t$. The particularity of topological measures is that they:

- require the graph structure to be computed,
- are holistic, thus, they need to be recomputed after each aggregation, i.e., the graph algorithm to compute the topological attributes has to be executed after each graph aggregation, instead of applying traditional aggregation functions. The topological dimensions, however, need the graph structure only at the initial phase to compute the base graph cuboid, and
- may need to be computed using a homogeneous subgraph of the multidimensional graph (most algorithms to compute topological properties such as centrality only make sense on homogeneous subgraphs).

Given a multidimensional graph, topological graph cubes are derived to capture the topological properties of graphs and represent them with numerical values (in contrast to traditional content-based properties that do not capture the topological characteristics).

Definition 5.11. [***Topological Graph Cube***] *A topological graph cube is a graph cube that captures the topological properties of graphs and represents them with numerical values. It is obtained by restructuring the topological multidimensional*


Star Schema
User Popularity Multidimensional Schema
Connected
Platform
User
Country
Community
Popularity
PageRank
Platform
name
Connected
Platform
User

Country
Platform
Community
PageRank
Traditional dimensional attribute
Traversal attribute
Topological dimensional attribute
Topological measure attribute
Traditional dimensional attribute

Fig. 5.5: Mapping Between OLAP Cube and a Multidimensional Topological Graph Schema

graph $\mathcal{G}_M = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H}, \mathcal{D}, \mathcal{M})$ *in all possible aggregations through the topological dimensions and/or by embedding and aggregating topological measures. That is,* $\exists \mathcal{D}^t \subseteq \mathcal{D} \parallel \exists \mathcal{M}^t \subseteq \mathcal{M}$.

The model we propose in this chapter could be mapped to a star schema shown in Figure 5.5. If the cube has a topological measure, we note that in order to guarantee a correct cube summarizability, roll up operations cannot be applied directly to OLAP cuboids to produce cuboids at higher aggregation levels, as typically done in OLAP. The reason is that following each roll-up operation, the graph structure changes and so do the topological values. The graph structure resulting cannot be expressed as a transitive function in terms of the input graph structure, as typically done in traditional OLAP cubes. Thus, topological graphs need to be computed once the cuboid they belong to has been created.

Example 3.1 (Popularity Graph Cubes)
Given a property graph, we suggest a process to enrich the graph with topological attributes, and derive a potential multidimensional schema and later its corresponding OLAP cube. We consider the example of a property graph representing a social network as in Figure 5.4. A single type of nodes and a single type of edges are considered: $\mathcal{L} = \{User, Connected\}$. This graph is enriched to capture the topological properties of users such as their community and PageRank. We design a possible multidimensional graph schema that embeds topological dimensions and measures. Three dimensions are considered: $\mathcal{D} = \{Country, Community, Platform\}$, and

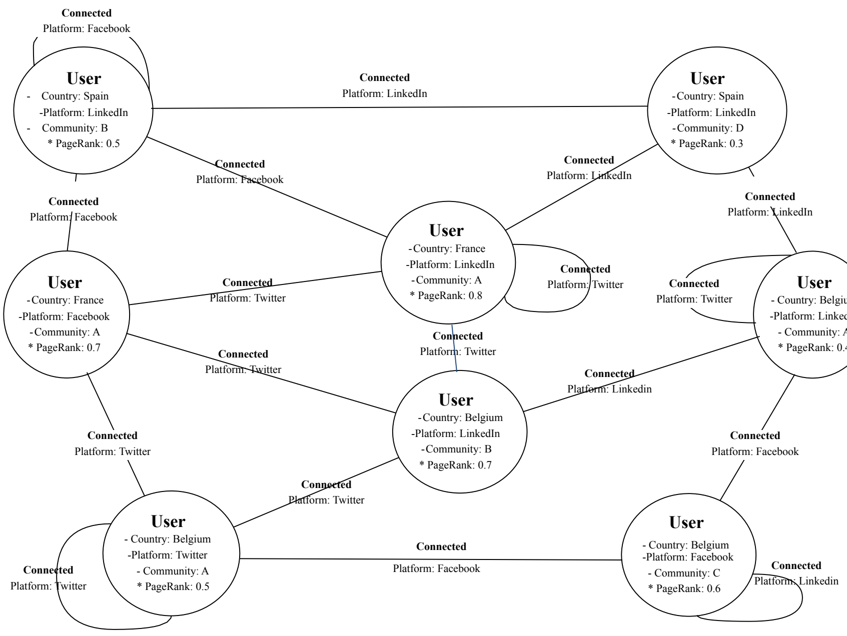


Fig. 5.6: Base Graph Cuboid Instance

the analyzed measure is *PageRank*. The measure is computed using the PageRank algorithm that traverses each time the edges that have the same value on their platform attribute (the traversal attribute). Figure 5.5 shows the mapping between the multidimensional popularity graph and its corresponding star schema. An instance of the popularity multidimensional graph at the base level is shown in Figure 5.6. Figure 5.7 shows an example of the popularity OLAP Cube corresponding to the graph cuboid of Figure 5.6, where community, platform, and country are the dimensions, and the PageRank is the topological measure.

3.2 Topological Graph Cuboid Processing

In this section we describe the topological cuboid aggregation algorithm 1. This algorithm is used to build topological graph cuboids. It takes as input a topological multidimensional graph, performs its aggregation along the given dimensions, and then applies the chosen aggregation function to compute the new measure values. The main phases of the algorithm are the following (note that this algorithm guarantees the multidimensional integrity

Algorithm 1: Topological Cuboid Aggregation

```
Input :
  • A topological multidimensional graph $\mathcal{G}_M = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H}, \mathcal{D}, \mathcal{M})$
  • Dimensions of the aggregate cuboid: $\mathcal{D}' \subset \mathcal{D}$
Output: An aggregate topological graph cuboid
        $\mathcal{G}'_M = (\mathcal{V}', \mathcal{E}', \rho, \mathcal{F}, \mathcal{H}, \mathcal{D}', \mathcal{M}')$
1  begin
2      Initialize a hash structure $\varphi : \mathcal{D}' \to \mathcal{V}' \cup \mathcal{E}'$
3      for $u \in \mathcal{V}$ do
4          if $\varphi(\mathcal{D}'(u)) = NULL$ then
5              Create an aggregate node $u' \in \mathcal{V}'$
6              $\mathcal{F}(u') \leftarrow \mathcal{F}(u)$
7              $\mathcal{D}'(u') \leftarrow \mathcal{D}'(u)$
8              $\mathcal{M}'(u') \leftarrow 0$
9              $\varphi(\mathcal{D}'(u)) \leftarrow u'$
10         for $M'_i \in \mathcal{M}'$ do
11             if $M'_i$ is NOT topological then
12                 $M'_i(u') \leftarrow compute(M'_i(u'), M_i(u))$
13     for $e(u, v) \in \mathcal{E}$ do
14         $u' \leftarrow \varphi(\mathcal{D}'(u))$
15         $v' \leftarrow \varphi(\mathcal{D}'(v))$
16         if $\varphi(\mathcal{D}'(e)) = NULL$ then
17             Create an aggregate edge $e'(u', v') \in \mathcal{E}'$
18             $\mathcal{F}(e') \leftarrow \mathcal{F}(e)$
19             $\mathcal{D}'(e') \leftarrow D'(e)$
20             $\mathcal{M}'(e') \leftarrow 0$
21             $\varphi(\mathcal{D}'(e)) \leftarrow e'$
22         for $M'_i \in \mathcal{M}'$ do
23             if $M'_i$ is NOT topological then
24                 $M'_i(e') \leftarrow compute(M'_i(e'), M_i(e))$
25     for $M'_i \in \mathcal{M}'$ do
26         if $M'_i$ is topological then
27             for $u' \in \mathcal{V}'$ do
28                 $M'_i(u') \leftarrow compute(u', M'_i, \mathcal{G}'_M)$
29             for $e' \in \mathcal{E}'$ do
30                 $M'_i(e') \leftarrow compute(e', M'_i, \mathcal{G}'_M)$
31     return $\mathcal{G}'_M = (\mathcal{V}', \mathcal{E}', \rho, \mathcal{F}, \mathcal{H}, \mathcal{D}', \mathcal{M}')$
```

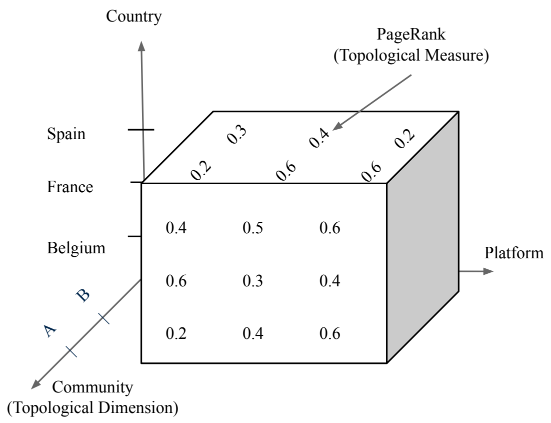


Fig. 5.7: Popularity Topological OLAP Cuboid

constraints discussed in Section 2.2):

1. Create a hash structure φ mapping each set of dimensional attributes from $\mathcal{D}'$ to an aggregate node/edge $u \in \mathcal{V}' \cup \mathcal{E}'$ (Line 2).

2. Create the set of aggregate nodes $\mathcal{V}'$: traverse the nodes of the multidimensional graph and create a node in $\mathcal{V}'$ corresponding to each subset of nodes in $\mathcal{V}$ sharing the same dimensional attribute values $\mathcal{D}'$ (Line 2-12). That is, for each node $u \in \mathcal{V}$ create its corresponding aggregate node $u' \in \mathcal{V}'$ (that has the dimensional attributes $\mathcal{D}'(u)$) if it was not already created in $\mathcal{V}'$ (Line 3-5). u and u' share the same label and dimensional attributes, and the measures are initialized (Line 6-8). The newly created node u' is stored as the value of the hash function corresponding to the dimensional attributes $\mathcal{D}'(u)$ (Line 9). Otherwise, if an aggregate node corresponding to u was already created, the non topological measure attributes $\mathcal{M}'(u')$ are updated using a user-defined function *compute* (Line 10-12).

3. Create the aggregate edges $\mathcal{E}'$: for each edge $e(u, v) \in \mathcal{E}$, we retrieve the aggregate nodes in $u', v' \in \mathcal{V}'$ corresponding to its adjacent nodes $u, v \in \mathcal{V}$ (Line 13-15). If $e'(u', v')$ was not yet created, then a new edge $e'(u', v') \in \mathcal{E}'$ is created (Line 16-17). e and e' share the same label and dimensional attributes, and the measures are initialized (Line 18-20). The newly created edge e' is stored as the value of the hash function corresponding to the dimensional attributes $\mathcal{D}'(e)$ (Line 21). Otherwise, if an aggregate node corresponding to e was already created, the non topological measure attributes $\mathcal{M}'(e')$ are updated using a user-defined function *compute* (Line 22-24).

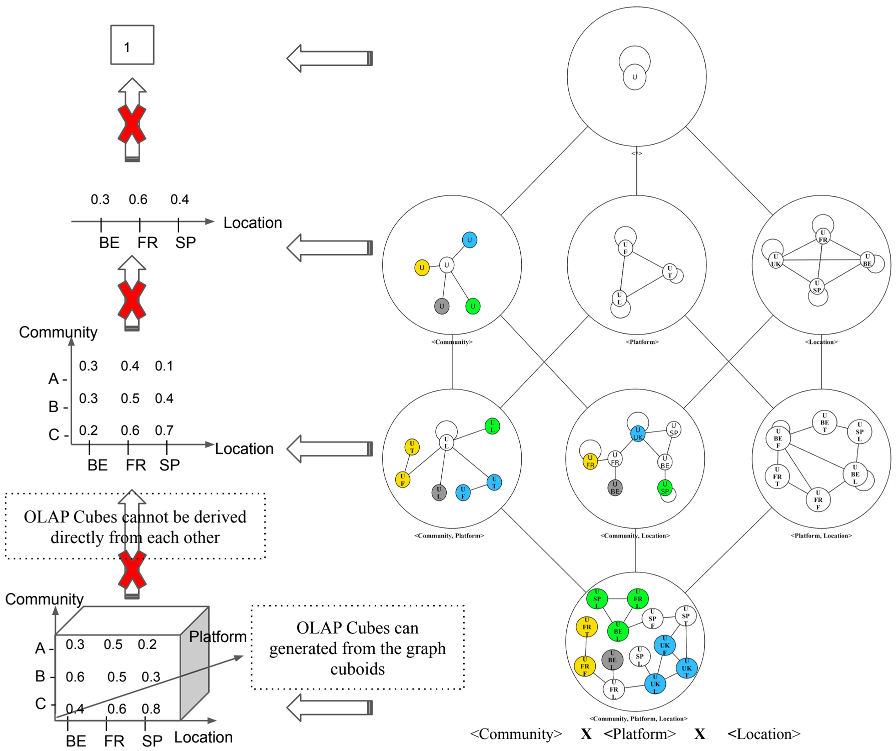


Fig. 5.8: OLAP Cube Generation form Graphs

4. If a measure M_i is topological, then it needs the whole aggregate graph $\mathcal{G}'_M$ to be built. The topological values for the nodes and edges are computed using the aggregate graph $\mathcal{G}'_M$ by applying a user-defined function *compute* (Line 25-30). The computation of topological values involves usually iterative graph algorithms that traverse the whole graph $\mathcal{G}'_M$.

3.3 Deriving OLAP Cubes from Graph Cubes

In this section, we detail our approach to derive OLAP cubes from multidimensional graphs, and precisely from their corresponding graph cubes. Most of the state-of-the-art techniques focus either on building traditional OLAP cubes, or building graph cubes. Here we propose to establish the link between the two. Thus, designing OLAP cubes that leverage the content and the topology of the graph, and expose both numerical and graph-structured insights.

The main current assumption is that each graph cuboid can be loaded into a relational OLAP cube. This is true for content-based graph cubes. However, loading a graph cuboid into a relational cube causes the loss of the graph structure. This loss of the graph structure has direct consequences for graph cube computation and analysis as follows:

- Graph cube computation: particularly roll up cannot be applied to get cuboids at higher lattice levels. The reason is that following each roll-up operation, the graph structure changes. Therefore, the topological measures on the aggregate graph need to be recomputed.
- Graph cube analysis: operations such as subgraph matching and traversal could no longer be executed on the extracted OLAP Cube. Therefore, as discussed in the previous section, the ability to deal with topological graph cubes is lost.

Example 3.2 (Deriving Popularity OLAP cube from the Graph cube)
Given the multidimensional graph model in Figure 5.4, we design a lattice, as shown in Figure 5.8. Each point in the lattice corresponds to a graph cuboid. For simplicity, we consider the dimension attributes: $\{Location, Community, Platform\}$, while ignoring the hierarchies of the location dimension. We highlight two particular aggregations: (1) node-only aggregations (i.e., only dimensional attributes from user nodes are kept not fully aggregated as in ($\langle Location, Community, *\rangle$, $\langle Location, *, *, *\rangle$, and $\langle *, Community, *\rangle$), and (2) edge-only aggregation as in ($\langle *, *, Platform\rangle$). The fact analyzed is the popularity of users. The measure is PageRank, computed by applying the PageRank algorithm in the social network following the edges labeled *Connected*.
The figure depicts the coupled processes of (1) aggregation of graph cubes, and (2) generation of corresponding OLAP cubes and the mapping kept between them. This mapping is important, as the graph topology corresponding to each OLAP cuboid needs to be preserved in order to compute the topological measures such as PageRank. The measures could afterward be loaded into the OLAP cubes for further multidimensional analysis.

The mapping discussed in this section could help in the integration of graph data and graph analytics within current data warehouses. However, the link is pretty limited when graph analytics are combined with OLAP. This has been the main assumption behind current graph OLAP tools, but this is not realistic given the relevance of graph-specific algorithms when dealing with graph data. Thus, given that most graph-derived cubes could not be supported with current relational warehousing systems, this motivates the need for building specialized graph OLAP warehousing systems.

4 Graph-structured Cubes

4.1 Graphs-structured Cube Model

Graph-structured cubes extend the traditional OLAP cubes with the capability of having the dimension and measure values represented as graphs. Current warehousing systems are not designed to support this type of cubes, which further motivates the need for developing native graph warehousing systems. In this section, we formally define the concepts of graph-structured dimensions, measures, and cubes.

The graph-structured dimensions are dimensions whose values are represented as graphs. They express complex dimension values that could not be represented by a simple value. This enables structuring the multidimensional space in a novel way capturing graph elements that are connected in a complex manner. Graph-structured dimensions provide therefore a powerful selection means to examine the behavior of non-trivial grouping of nodes or edges.

The definition of graph-structured dimensions relies on graph patterns defined as follows:

Definition 5.12. [***Graph Pattern***] *A graph pattern $\mathcal{P}$, over a property graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \rho, \mathcal{F}, \mathcal{H})$, is defined as $\mathcal{P} = (V_p, E_p, \alpha, \beta)$, where:*

- *V_P is a finite set of nodes.*
- *E_P is a finite set of edges.*
- *α is a function defined on $V_P \cup E_P$ such that for each node $u \in V_P$ (resp., edge $e \in E_P$), $\alpha(u)$ (resp., $\alpha(e)$) is the predicate applied on the label of u (resp., e). This predicate compares the label $l_i = \mathcal{F}(u)$ of u (resp., of e) with a string s_j. The comparison is of the form l_i op s_j, and is performed using one of the two equality comparison operators $=, \neq$.*
- *β : is a function defined on $V_P \cup E_P$ such that for each node $u \in V_P$ (resp., edge $e \in E_P$) , $\beta(u)$ (resp., $\beta(e)$) is the predicate applied on the attributes of u (resp., e). This predicate is a conjunction of atomic formulas each of which compares a constant c with the value x of an attribute of u (resp., e) using a given operator op_i. The comparison is performed using any of the following operators: $<, \leq, =, \neq, >, \geq$. Hence, $\beta(u)$ (resp., $\beta(e)$) is a conjunction of comparisons of the form: $c\ op_i\ x$.*

Definition 5.13. [***Graph-structured dimension***] *A graph-structured dimension D_i is a dimension represented with a graph pattern $\mathcal{P}$ used for selecting a subset of the graph elements. The set of graph-structured dimensions is $\mathcal{D}^s =$*

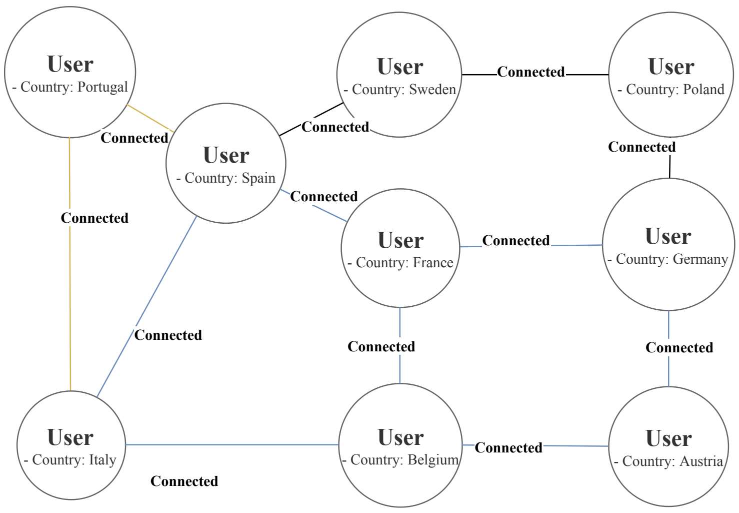


Fig. 5.9: Graph-structured Cuboid

$\{D_1{}^s, D_2{}^s, ..., D_n{}^s\} \subseteq \mathcal{D}$. *Each graph-structured dimension* $D_i{}^s$ *is represented by a graph pattern* $\mathcal{P}_i$.

The graph-structured measures are measures where the values are represented as graphs, which enables capturing and exposing insights and metrics structured as graphs. Another main benefit of graph-structured measures is that they minimize the information loss, as they keep the graph structure after being computed or aggregated.

Definition 5.14. [***Graph-structured Measure***] *A graph-structured measure* $M_i{}^s$ *is represented with a subgraph* $\mathcal{G}^s$. *It is computed using a graph function* Δ *that takes a graph as input and returns a graph, such as the most frequent pattern, or minimum spanning tree. An aggregation function* ϕ *is used to compute the graph-structured measure at different aggregation levels such as intersection or union of graphs.* ϕ *and* Δ *could be the same function (thereby recomputing the measure after each aggregation).* $\mathcal{M}^s = \{M_1{}^s, M_2{}^s, ..., M_n{}^s\} \subseteq \mathcal{M}$

Definition 5.15. [***Graph-structured Cube***] *A graph-structured cube is a graph cube that captures and represents the dimensional concepts using graphs. Therefore, it contains either graph-structured dimensions or graph-structured measures, or both. A graph-structured cube is obtained by restructuring the multidimensional graph in all possible aggregations through the graph-structured dimensions and/or by embedding and aggregating graph-structured measures.*

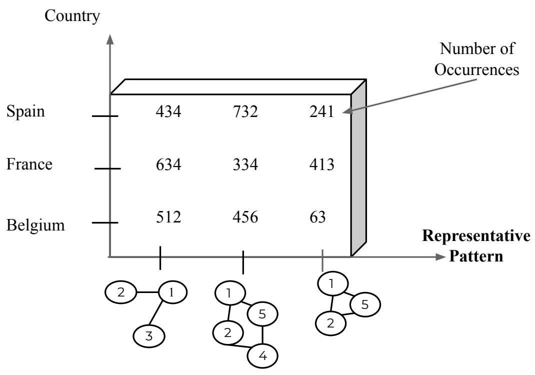


Fig. 5.10: Graph-structured Dimension

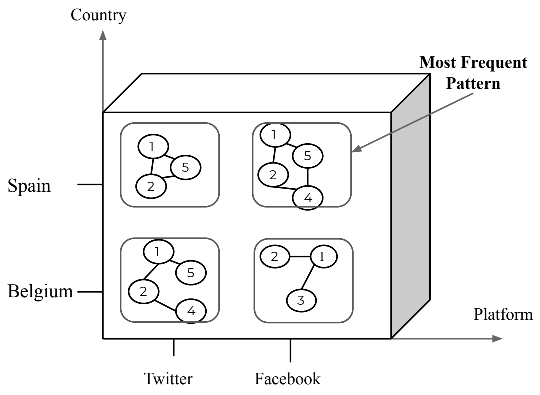


Fig. 5.11: Graph-structured Measure

In the following example, we illustrate two graph-structured cubes, highlighting respectively graph-structured dimensions and measures. For each cube, we show how graph-structured multidimensional structures could be combined with the numerical ones defined in the previous sections.

Example 4.1
(Graph-structured Cubes) Consider the graph cuboid of Figure 5.9. It represents a graph cuboid where users are grouped using the country dimension. Figure 5.10 shows a cube that highlights the case of graph-structured dimensions. For this example, we assume that we can extract a set of patterns that represent the graph elements, and we call these patterns the representative patterns. The horizontal axis of the cube is then populated by a set of graph patterns depicting the representative patterns. Using this cube, the user can analyze for example how often users from a given country are involved in a representative pattern. Those are complex dimension values that could not be represented by a numerical value and need therefore to be defined by patterns. To define the dimension values, the user could either find the pattern using graph algorithms or use his domain knowledge. The three patterns of the graph-structured dimension of the cube of Figure 5.10 are represented on the graph cuboid Figure 5.9 using different colors for each. Figure 5.11, on the other hand, puts the focus on graph-structured measures. The measure studied here is the most frequent pattern, which could be obtained by applying the graph algorithms. Each measure (i.e., frequent pattern) is then placed within the graph-structured cube using two traditional dimensions: country and platform. This cube could be used to analyze the most frequent pattern observed by country and platform.

4.2 Graph-structured Cuboid Processing

Graph-structured cuboid aggregation is similar to topological cuboids' aggregation. It is performed along the lines of Algorithm 1, while adapting the selection and aggregation to handle the graph patterns. It takes as input a multidimensional graph and performs its aggregation using a given set of dimensions, then applies a chosen graph aggregation function to compute the new measure values. Considering a cube with a set of graph-structured dimension and measures, the main steps of the algorithm are:

1. Create a hash structure $\varphi : \mathcal{P}' \rightarrow \mathcal{V}' \cup \mathcal{E}'$, mapping each pattern (corresponding to a graph-structured dimension) to an aggregate node/edge.

2. Create the set of aggregate nodes $\mathcal{V}'$: traverse the nodes of the multidimensional graph and create a node $u'_i \in \mathcal{V}'$ corresponding to the subset of nodes in $G_u = \{u_1, ..., u_n\} \subseteq \mathcal{V}$. G_u is the set matching the pattern P'_i representing a dimensional value of D'_i. The aggregate node $u'_i \in \mathcal{V}'$ is only created if no nodes in $\mathcal{V}'$ corresponding to P'_i were already created. u and u' share the same label and non-structured dimensional attributes, and it gets assigned its graph-structured dimension D'_i. The newly created node u' is stored as the value of the hash function corresponding to the pattern P'_i. Otherwise, if an aggregate node corresponding to u was already created, the non-topological measure attributes $\mathcal{M}'(u')$ are updated.

3. Create the aggregate edges $\mathcal{E}'$: this step is similar to the one in Algorithm 1. That is, for each edge $e(u, v) \in \mathcal{E}$, if no corresponding edge $e'(u', v') \in \mathcal{E}'$ was created, e' is created and inherits the same label and dimensional attributes of e. The aggregate edge e' is stored as the value of the hash function corresponding to P'_i. Otherwise, the non-topological measure attributes $\mathcal{M}'(e')$ are updated.

4. Topological and graph-structured measures are computed for the aggregate graph.

The description of the algorithm focuses on the matching of a single dimensional value. Given all the dimensions that could be considered in a given graph cube, for each cell the graph-structured measure is the subgraph that matches the patterns of all the dimensions (i.e., the intersection of the subgraphs that match each pattern).

5 OLAP Analysis of Graph Cubes

OLAP analytics supports interactive and complex queries over large volumes of data, from different perspectives and through different hierarchical levels. Thus, enabling analysts to highlight the data item of interest, and then drill down to the underlying data from which it has been created. This could help in decision support scenarios such as the measurement or comparison of the business performance across the different dimensions. In this section, we describe a set of algebraic operators for OLAP querying of multidimensional graphs. We consider the graph cubes defined and computed in the previous sections as the fundamental construct of the multidimensional model. The graph cubes are the operand and the return type of all OLAP operations. We illustrate the application of each operation on a graph cuboid and its corresponding OLAP cube. In addition to the cuboid and crossboid operations that were defined in the literature [143], we present the major OLAP operations applied on graph and OLAP cubes.

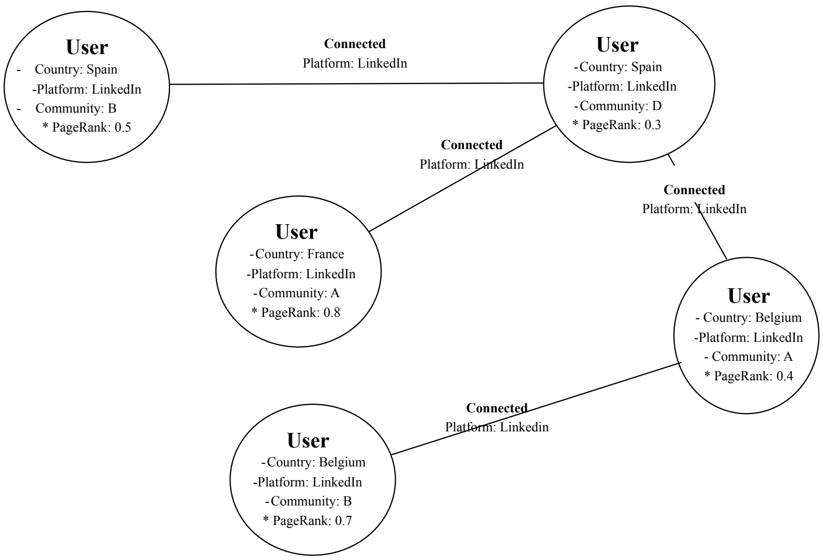


Fig. 5.12: Slice on the LinkedIn Platform Dimension

Multidimensional Selection Multidimensional selection (also called a slice) (denoted as $\sigma_{\mathcal{P}}(\mathcal{G}_M)$ restricts the graph $\mathcal{G}_M$ to a subgraph $\mathcal{G}'_M \subseteq \mathcal{G}_M$ where all nodes and edges match the selection pattern $\mathcal{P}$. The selection pattern could be a conjunction of (1) atomic predicates applied to one or more dimension attributes in the case of property and topological cubes, or (2) graph patterns in the case of graph-structured cubes, or (3) a combination of both. The result $\mathcal{G}'_M$) is a set of nodes and edges that are matched by the selection pattern $\mathcal{P}$. The algebra of the selection operator is defined as follows:

- Input: A graph cuboid $\mathcal{G}_M$ and a selection pattern $\mathcal{P}$.
- Output: A graph cuboid $\mathcal{G}'_M \subseteq \mathcal{G}_M$, that matches the selection pattern $\mathcal{P}$.
- Example: The result of a selection applied on the graph cuboid of Figure 5.6 is shown on Figure 5.12, where only user nodes of the LinkedIn platform are selected.

Roll-up and Drill-down Roll-up (denoted as $\mathcal{R}_{D_i}(\mathcal{G}_M)$) aggregates the graph $\mathcal{G}_M$ along the dimension D_i. The graph is either aggregated to the next dimension hierarchy level if D_i is part of a dimension hierarchy following the partial order $\mathcal{R}$, or to ALL. This operation modifies the granularity

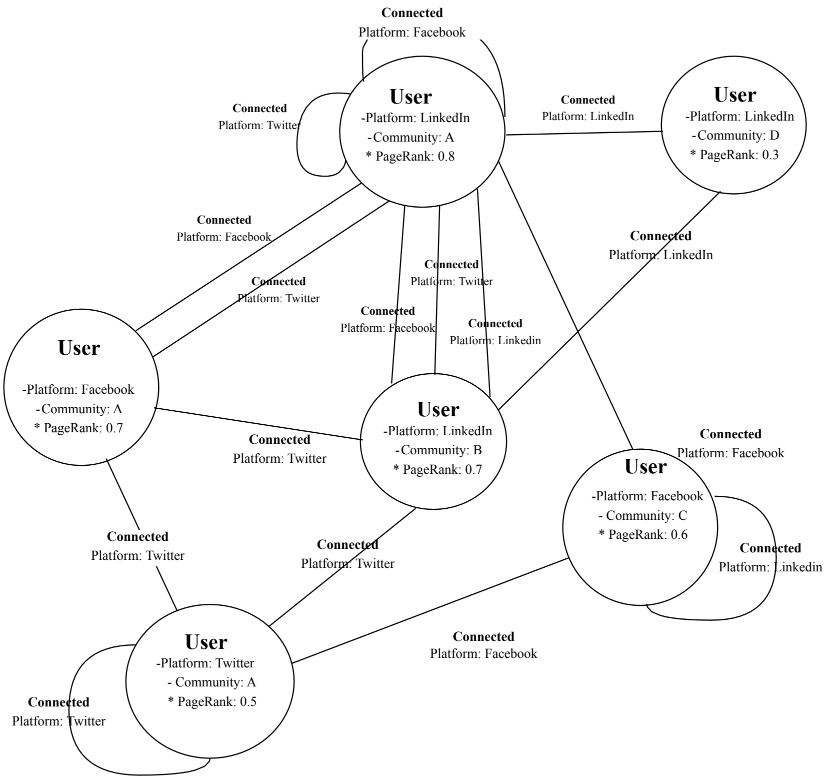


Fig. 5.13: Popularity Graph Cuboid Rolled up to $\langle *, Community, Platform\rangle$

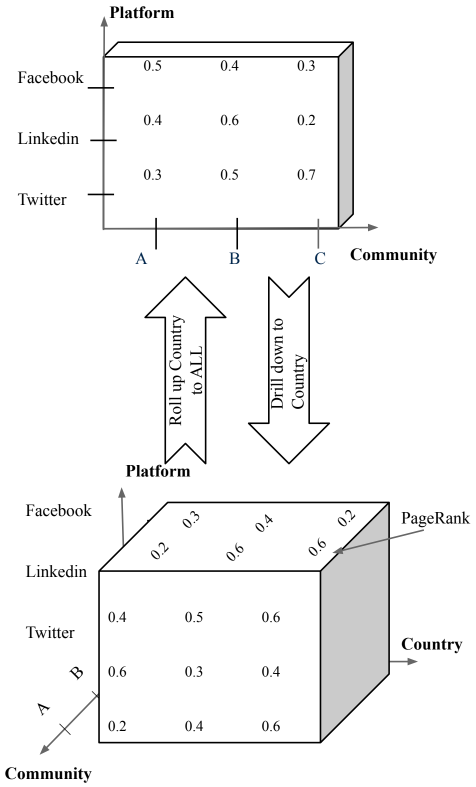


Fig. 5.14: Roll-up and Drill-down on the Popularity OLAP Cube

of the graph using a many-to-one relationship which relates instances of two levels in the same dimension hierarchy, corresponding to a part-whole relationship.
This operation performs structural changes to the graph and generates a new graph placed at the next level of the dimension hierarchy while respecting the summarizability integrity constraints. Roll-up is implemented in three phases (1) first a selection of graph elements matching the aggregation pattern $\mathcal{P}_{agg}$ that describes the graph elements at $Level_i$, then, (2) the graph aggregation to shape the graph at $Level_{i+1}$, and finally, (3) measures are (re)computed and placed on the aggregate graph. The algebra of the roll-up operator is defined as follows:

- Input: Initial graph cuboid: $\mathcal{G}_M$; The dimension to aggregate D_i.
- Output: A graph cuboid $\mathcal{G}'_M$. All elements of the initial graph cuboid $\mathcal{G}_M$ that contain the dimensional attributes of D_i are grouped in their corresponding node (resp. edge). The measure values on the aggregate nodes and edges are computed according to their aggregation function ϕ.
- Example: The result of a roll-up $Country \rightarrow ALL$ applied on the graph cuboid of Figure 5.6 is shown on Figure 5.13, where user nodes are grouped by community and platform, and the new page rank value is computed for each node. Figure 5.14 shows the equivalent roll-up and drill down operations applied on the corresponding OLAP cubes.

Roll-up is similar to the *cuboid* operation defined in the GraphCube paper [143]. Drill-down is the inverse of roll-up, and can only be applied if we previously performed a roll-up and did not lose the correspondences between the graphs.

Drill-across and Projection This operation changes the subject of analysis of the cube using a one-to-one relationship. The n-dimensional space remains the same, only the cells placed on it change. With this operation, different measures are placed on the same multidimensional space. This operation translates to a join between two graph cuboids put on the same multidimensional space, at the same aggregation level. The join condition for nodes could be their identifier attributes. Projection is the reverse operation of a drill-across. It selects a subset of measures of interest to be studied within the multidimensional space. The algebra of the drill-across operator is defined as follows:

- Input: Initial graph cuboids: G^1, G^2, and the measures m_i, m_j.
- Output: A graph cuboid G^3, union of G^1, G^2, where the concerned graph elements embed both measures m_i and m_j.

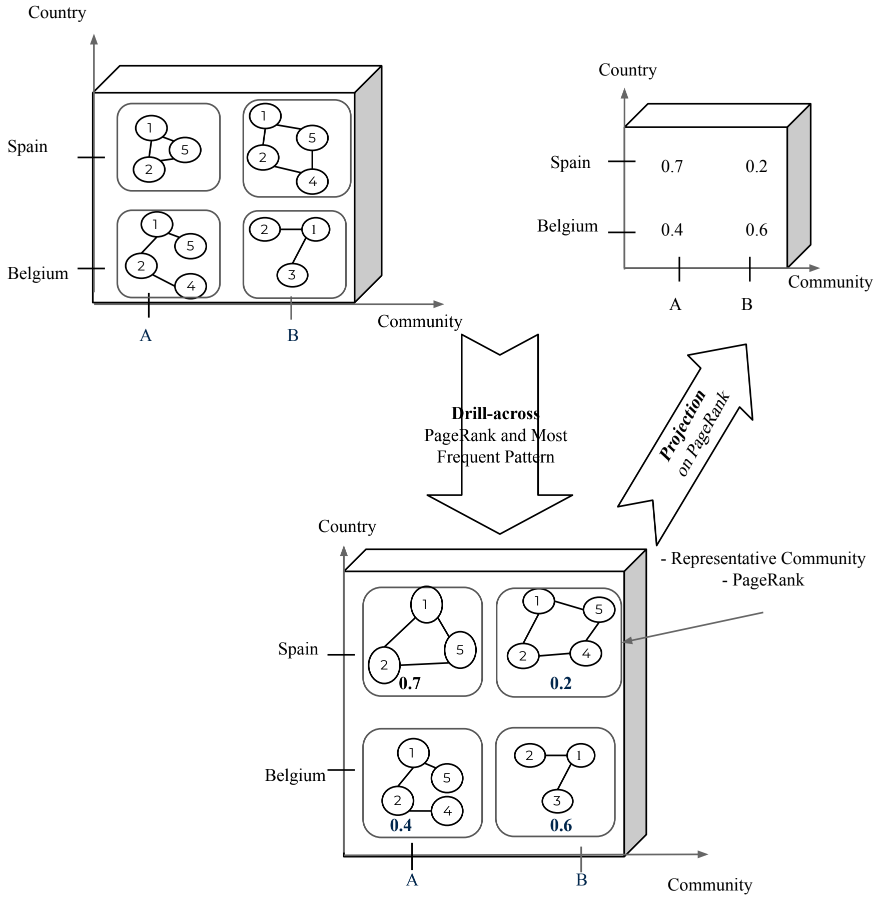


Fig. 5.15: Drill-across and Projection

- Example: Figure 5.15 shows an example of a drill-across between a topological and a graph-structured cube. Both cubes are placed in a cube having as dimensions ($\langle Community \rangle$, $\langle Country \rangle$). The first is a graph-structured cube containing representative communities, and the second is a topological cube containing PageRank. Using drill-across, the measures from the two cubes could be embedded in the same cells and analyzed within the same cube. Inversely, a projection would for instance remove the measure representative community from the cube to focus only on studying the PageRank.

In the same way, further OLAP operators could be applied to the graph cubes for richer or more intuitive analysis. For example, the difference between graphs removes isomorphic subgraphs that exist in the two input graphs. Drill-through enables direct access to the subgraph that was initially used for the computation of the cube's measures. It goes beyond drill-down to explore the lowest aggregation level present in the physical graph, and non-necessarily reached at the data mart level. In general, this chapter opens the door to advanced operators combining graph-like and OLAP operators.

6 Implementation and Experiments

Current decision-support systems, and particularly data warehouses, were designed to support relational data management and analysis. Due to the fundamental difference between graph and relational data, the existing systems are not suitable for efficient graph analysis. The structure-driven management and analytics of graph data call for rethinking the architecture of data warehouses to support graph analytics, and to the development of novel data models, query processing paradigms, and storage techniques.

6.1 Framework Architecture and Implementation

The architecture of the graph warehousing and analysis framework is depicted in Figure 5.16. To exemplify it, we use the same running example and take Twitter as source data. The modules are described as follows:

- Graph Extraction: Graph data is extracted from the source. In our running example, using the Twitter streaming API. A set of transformations is then applied to cleanse the data and fit it within the envisioned schema. The stream is parsed to identify the data entities and merge duplicates and compute new attributes such as length of tweets and their sentiment. For this purpose, any generic tool would suffice.

- Graph Construction: The clean data is loaded in the graph store. In this case, we used Neo4j to store the graph data and Cypher queries to perform the loading. The cleansed and integrated Twitter data is therefore natively stored and managed as a multidimensional graph.

- Graph Cube Construction: Multiple multidimensional schemas could be built from the same graph warehouse to satisfy the various analysis needs. The semantic relativism inherent in graphs allows creating several views from the same data and making them co-exist in a much simpler way than any other data model. Therefore, given a graph lattice, the graph cube framework enables the computation and the aggregation of the corresponding graph cuboids. Each graph cuboid is computed and persisted in a graph store that resembles a graph mart.

The graph cuboid stores natively different graph measures (e.g., centrality, shortest paths, frequent patterns, etc.). An example of the graph cuboid computation is shown below.

- Graph Analysis: Complex and interactive analysis of graph cubes is performed at this phase. In contrast to traditional OLAP analytics, graph analytics enables BI-oriented analysis of graph metrics stored in the graph cuboids. For example, analysts could examine at different levels of aggregation and from multiple perspectives graph measures such as influence (e.g., computed using centrality), or identifying communities and their connections (e.g., computed using graph clustering). Importantly, note that traditional visualization tools do not suffice to deal with interactive graph analysis, especially graph-structured cubes. Therefore, an ad-hoc graph browser was implemented.

We implemented the architecture described above as a prototype graph warehousing system. We used Neo4j for graph data management and Neo4j graph algorithms and JUNG (Java Universal Network/Graph Framework) for graph mining. The Java code below shows an example of cuboid computation to perform an aggregation on the dimensional attribute: *sentiment*. Here, we specify the input dataset and output directory, the dimensions and their dimensional attributes, the measures, and their computation and aggregation functions. The following code illustrates our API:

```
//Graph Cuboid Builder
GraphAggregator graphAggregator = GraphAggregator.builder()
//Input: Multidimensional Graph
.basePath(Paths.get("data/MDTwitter"))
//Output: Graph Cuboids
.workPath(Paths.get("data/TwitterCuboid"))
//Vertices
.vertex("User")
//Vertex dimensions
.dimension("Date", DimensionAgg.KEEP)
.dimension("Language", DimensionAgg.KEEP)
//Traditional vertex measure
.measure("followers", "TotalFollowers",
AggFunction.COUNT)
.vertex("Tweet")
//Vertex dimensions
.dimension("Language", DimensionAgg.KEEP)
.dimension("Sentiment", DimensionAgg.IGNORE)
//Topological vertex measure
.structuralMeasure(StructuralMeasure.SM.LOUVAIN,
"RETWEETED")
//Edges
```

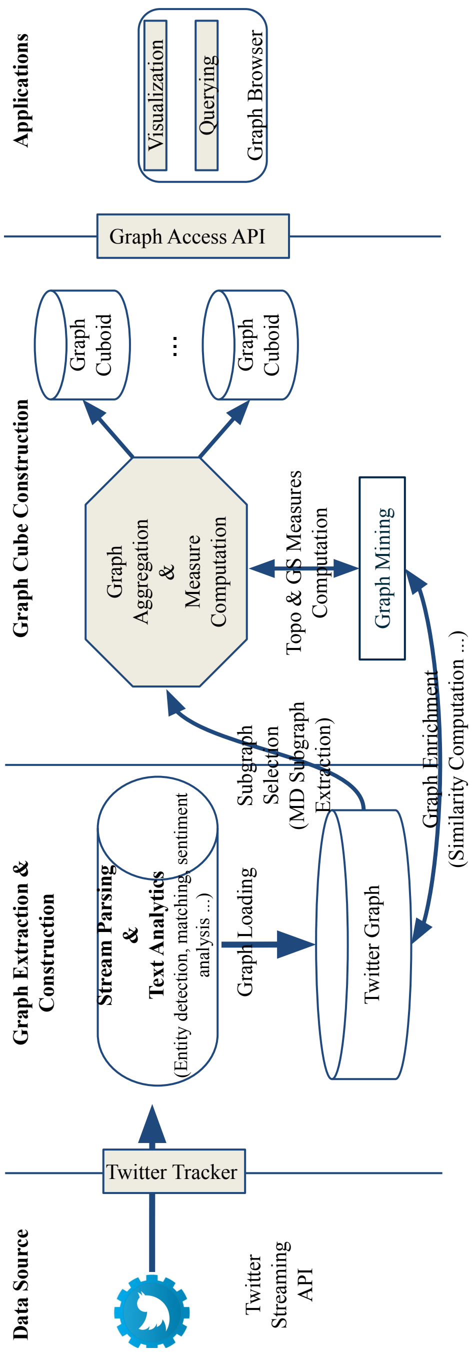

Data Source
Twitter Streaming API
Twitter Tracker
Graph Extraction & Construction
Stream Parsing & Text Analytics (Entity detection, matching, sentiment analysis ...)
Graph Loading
Twitter Graph
Subgraph Selection (MD Subgraph Extraction)
Graph Enrichment (Similarity Computation ...)
Graph Cube Construction
Graph Aggregation & Measure Computation
Topo & GS Measures Computation
Graph Mining
Graph Cuboid
...
Graph Cuboid
Graph Access API
Applications
Visualization
Querying
Graph Browser

Fig. 5.16: Twitter Network Warehousing Architecture

```
.edge("POSTED")
.edge("MENTIONED")
.edge("REPLIED_TO")
.edge("RETWEETED")
.build();
graphAggregator.aggregate();
```

6.2 Experiments

In this section, we present the experimental results of our graph OLAP framework using multiple real-world datasets. We compare the cuboid generation and aggregation time for each dataset at different aggregation levels.

Datasets We ran the experiments on two types of real-world datasets. The first are three Twitter datasets, of size 500K, 1M, and 2M edges. The data is collected using Twitter streaming API as depicted by the framework of Figure 5.16 described above. The original stream contained two types of nodes: *User and Tweet*, and four types of edges: *POSTED, RETWEETED, MENTIONED, and REPLIED_TO*. We enriched the Tweet nodes by computing the sentiment of the tweets. Table 5.1 provides a summary of the characteristics of the multidimensional social network built using the Twitter datasets. The code to build Twitter cuboids was shown in the previous subsection.

	Dimensional Attributes	Measures
User	Language, Subscription Date	Number of Followers, Number of persons
Tweet	Language, Sentiment	Number of tweets, Community
Edges	none	Number of edges

Table 5.1: Twitter Datasets

The second type of graphs uses four datasets from the SNAP collection [81]. The original dataset contains only the graph structure between users. We use this dataset to experiment with the computation and aggregation of topological dimensions and measures of the multidimensional graph. For the nodes, we computed three topological properties that we considered as dimensions (PageRank, triangles, and clustering coefficient). For the measures, we computed the community by label propagation and considered it as a measure for the nodes. We consider the count of nodes and edges as a measure each time we aggregate the graph. The ability to derive new multidimensional measures and dimensions using only the graph structure shows an interesting aspect of graphs and the potential of multidimensional graph analytics, even when we have no content-related information about the original data. Table 5.2 shows the evolution of the graph order and size through consecutive multidimensional aggregations of the graph. Since the graph is homogeneous, we end up always with a single node and edge that summarizes the graph at the apex level.

Dataset	Original		Base		TR-PR		PR		Apex	
	#V	#E	#V	#E	#V	#E	#V	#E	#V	#E
DBLP	317,080	1,049,866	40,598	869,814	13,926	741,466	275	16,552	1	1
Youtube	1,134,890	2,987,624	41,704	1,835,365	15,067	1,459,786	652	51,840	1	1
Skitter	1,696,415	11,095,298	164,462	7,865,009	55,430	5,994,331	1,176	119,731	1	1
LiveJournal	3,997,962	34,681,189	564,648	33,382,661	122,331	28,683,757	700	70,480	1	1

Table 5.2: Graph Cuboids Order and Size

To build the graph cuboid for the SNAP graphs, we use the following code:

```
//Graph Cuboid Builder
graphAggregator = GraphAggregator.builder()
.basePath(Paths.get(DB_PATH))
.workPath(Paths.get(DB_PATH + "_aggregatedbase"))
.vertex(node)
.dimension("pagerank", DimensionAggregation.KEEP)
.dimension("coefficient", DimensionAggregation.KEEP)
.dimension("triangles", DimensionAggregation.KEEP)
.structuralMeasure(StructuralMeasure.SM.LABEL_PROP, edge)
.edge(edge)
.build();
```

Framework Efficiency The graph extraction and construction algorithms and the experimental setup were implemented in Java. For the first type of datasets, the framework was tested on a single machine with 16 GB of RAM, and an Intel(R) Core(TM) i5-7200U CPU@2.50GHz, running on Ubuntu 18.04. For the second type, with larger datasets, we used a machine with 256 GB of RAM, and an Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz, running on Ubuntu 18.04. The proposed system uses the centralized graph database Neo4j. The graphs were implemented using adjacency lists as it is a more compact representation. For the processing, hashmaps are used as described in the algorithm.

Given that the Twitter dataset is a heterogeneous graph, with multiple types of nodes and edges, we compute the time to build the cuboid at the base and apex level, and at the end two aggregate cuboids that aggregate the tweets(Tweet-Agg) and users(User-Agg) respectively. For the base level, Base-C refers to cuboid computation with content measures only, while Base-T refers to cuboid that has both topological and content measures. Figure 5.17 shows the computation time with the JVM Xms and Xmx set to 8 GB. Figure 5.18 shows the aggregation of the SNAP networks. Given the raw datasets, first, the multidimensional graph is computed, then aggregated

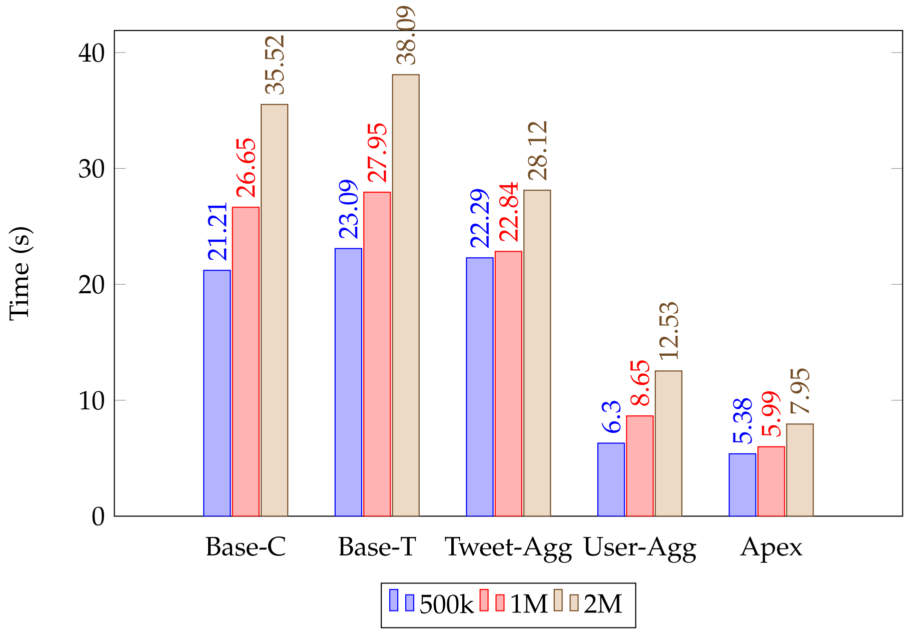


Fig. 5.17: Computation Time for Building the Graph Cuboids

through different dimensional levels. We run the experiment on the machine with 256 GB, but we set the JVM Xms and Xmx to 32 GB, except for the LiveJournal aggregations where we encounter an out of memory error. The results in Figure 5.18 show the computation time for the different graph cuboids. MD refers to the computation of the multidimensional graph, given the raw input from SNAP. Base refers to the base graph cuboid, PR-TR is the cuboid where the coefficient is aggregated, and PR is the cuboid where the graph is aggregated on both coefficient and triangles dimensional attributes, and Apex to the highest aggregation level.

Following these experiments, we notice that the processing time depends on the size, order, and volume of the input and output graphs. The order of the graph is the number of its nodes and size refers to the number of its edges. As we consider both content and structural information present in the graph, all these properties have a direct effect on the efficiency of the aggregation. Given that at the first aggregation level we have many possible dimension values for the dimensional attributes, we end up with a graph close in size and order to the base graph, therefore exhibiting similar computation time. This explains, for example, why User-agg that aggregates users is faster than Twitter-agg. We also notice that most of the computation time is spent on the two phases: the I/O phase, where the graph is loaded from and to the disk, then the graph aggregation phase, where the nodes and edges are merged.

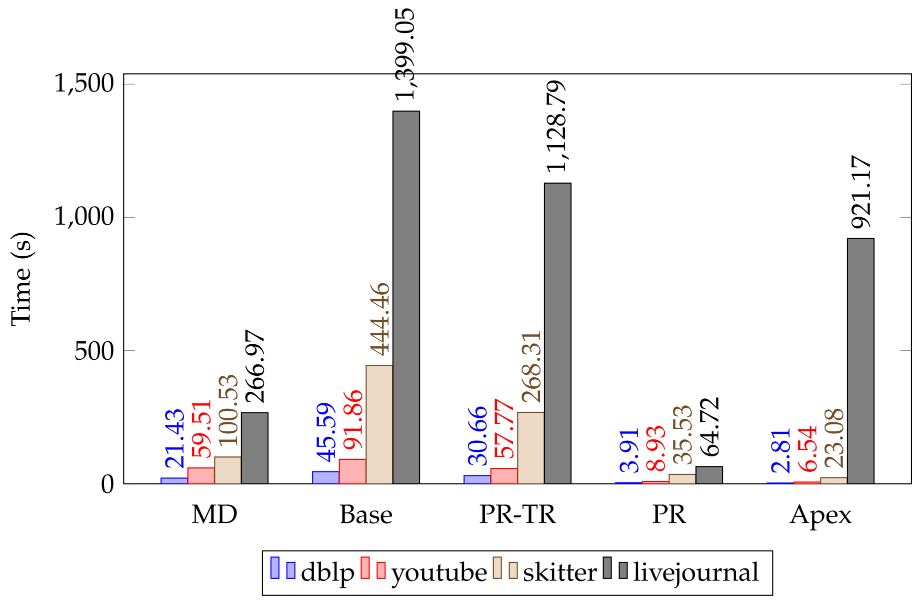


Fig. 5.18: Cuboid Aggregation Time

The overhead of computing topological measures is very small as shown in Figure 5.17. This is due in part to the fact that the graph algorithms are executed within the database engine, without losing I/O to export the graph to a processing library then import it again. Therefore, we got performance orders of magnitude better than those when we used an external generic Java graph library such as JUNG and jGraphT.

7 Conclusion and Open Challenges

In this chapter, we extended the state of the art on graph warehousing by designing a multidimensional graph model that leverages the content and the topology of the graph. We proposed for the first time a model that exposes both numerical and graph-structured insights using graph cubes while preserving multidimensional integrity constraints. Furthermore, we proposed different analytical scenarios and formalized the OLAP querying of the graph cubes. We also discussed the potential correspondence between graph cubes and traditional ROLAP cubes. Concerning the implementation, we have detailed the framework architecture and the system API and evaluated its efficiency with multiple real-world datasets.

As a future research direction, our target is to improve the performance of TopoGraph using a distributed graph engine and combine existing graph cube materialization techniques with our novel cube definitions. We also plan to extend our work on dynamic graphs to support continuous updates and analysis of real-time graph data. Further work needs to be done on defining a multidimensional query language for graphs and designing efficient optimization strategies. Machine learning algorithms could also be used to enable advanced mining scenarios such as the discovery of interesting patterns in the graph cube, and the prediction of the graph cube evolution.

Chapter 6

Graph BI & Analytics: Current State and Future Challenges

The paper has been published in the *Proceedings of the 20th International Conference on Big Data Analytics and Knowledge Discovery. (DaWaK 2018).*
The layout and content of the paper have been revised.
DOI: https://doi.org/10.1007/978-3-319-98539-8_1

Abstract

In an increasingly competitive market, making well-informed decisions requires the analysis of a wide range of heterogeneous, large, and complex data. This paper focuses on the emerging field of graph warehousing. Graphs are widespread structures that yield great expressive power. They are used for modeling highly complex and interconnected domains, and efficiently solving emerging big data application. This paper presents the current status and open challenges of graph BI and analytics and motivates the need for new warehousing frameworks aware of the topological nature of graphs. We survey the topics of graph modeling, management, processing, and analysis in graph warehouses. Then we conclude by discussing future research directions and positioning them within a unified architecture of a graph BI and analytics

framework.

1 Introduction

Graphs are fundamental and widespread structures that provide an intuitive abstraction for the modeling and analysis of complex, heterogeneous, and highly interconnected data. They have the benefit of revealing valuable insights from content-based and topological properties of data. The great expressive power of graphs, along with their solid mathematical background, encourages their use for modeling domains having complex structural relationships. In the context of Big Data, the focus of organizations is often on handling the rising volume of their data. However, the variety and complexity of data through the different phases of data capturing, modeling, and analysis are at least equally important. The variety challenge is the most critical in big data nowadays, and efficiently handling the variety of data sources is considered to be the main driver of success for data-driven organizations [15]. Graphs meet the requirements to be the perfect canonical data model for data integration systems [47] given (1) their capability to deal with semantic relativism and semantic heterogeneities, (2) they are semantically richer at least as any other model (so they can represent any semantics), (3) they allow to create multiple views from the same source, (4) and most importantly, graphs are extremely flexible to compose new graphs. That is given two graphs, with one single edge a new graph could be directly created without affecting the existing ones. Therefore, graphs are suitable to deal with big data variety better than any other data model.

Furthermore, in industry, graph analysis is considered as *"possibly the single most effective competitive differentiator for organizations pursuing data-driven operations and decisions after the design of data capture"* according to Gartner, Inc., a research and advisory firm [45]. Indeed, large complex graphs have emerged in various fields and graph analytics are being increasingly used to solve complex real-world problems. In the financial sector, for example, several types of fraud could be detected and prevented in auction and transaction networks [4]. In [130], the authors used bank transactions to build a financial transactions network, where each node represents a client, and each edge represents a transaction. As fraudsters tend to collaborate to orchestrate complex fraud at large scale, the probability that a customer is involved in a fraud depends on his neighborhood in the transaction graph. Graph analytics could be used to define and retrieve complex fraud patterns, or to score customers by fraud exposure. In [40], the authors built a Call Detail Record graph to understand the call routines and interactions between customers. This information can later be used to prepare marketing campaigns or to prevent customer churn.

The topological properties of graphs are of big potential to decision-making systems. They supply these systems with a new class of complex structural business facts and measures that could be explored for making a more accurate decision in data-driven organizations. In current information systems, Business Intelligence (BI) systems are critical for strategic decision making. Graph BI, in particular, is emerging as the BI field that extends current BI systems with graph analytics capabilities. It enables graph-based insights such as detection of popular users or communities in social networks, or revealing hidden interaction patterns in financial networks. Graph BI can help address the above-mentioned big data applications since (1) data are interconnected in complex ways, but graphs can help reduce this complexity with intuitive data models and queries, (2) the data size is large, but data warehouses and Online Analytical Processing (OLAP) analysis are suitable for storage, organization, synthesis and analysis of large volumes of data, and (3) graph mining extends traditional techniques by including the discovery of the topological properties, thus characterizing more precisely business applications. Traditional BI systems, and particularly data warehouses, were designed to support relational data management and analysis. Due to the fundamental difference between graph and relational data, the existing systems are not suitable for efficient graph analysis.
The structure-driven management and analytics of graph data call for the development of novel data models, query processing paradigms, and storage techniques. Therefore, as motivated by multiple research lines [43, 83], current BI and analytics systems need to be extended to efficiently support warehousing [38], processing [120], mining [119] and OLAP analysis [28, 58] of the graph structural and content-based information.

Figure 6.1 provides an overview of the different components of the envisioned graph BI system. While adopting a similar template as the traditional BI systems (i.e., it preserves the familiar data analytics workflow), graph BI extends current systems with graph-aware components that deliver graph-derived insights.

Note that through this paper the terms "graph and network", "node and vertex", and "edge and relationship" are often used interchangeably. The remainder of this paper presents the current state and the open challenges of graph BI & analytics, with a focus on graph warehousing. Section 2 discusses the topic of graph data modeling and management. Section 3 surveys the existing frameworks for graph analytics. Section 4 identifies future research directions and position them within a unified architecture of a graph BI & analytics framework.

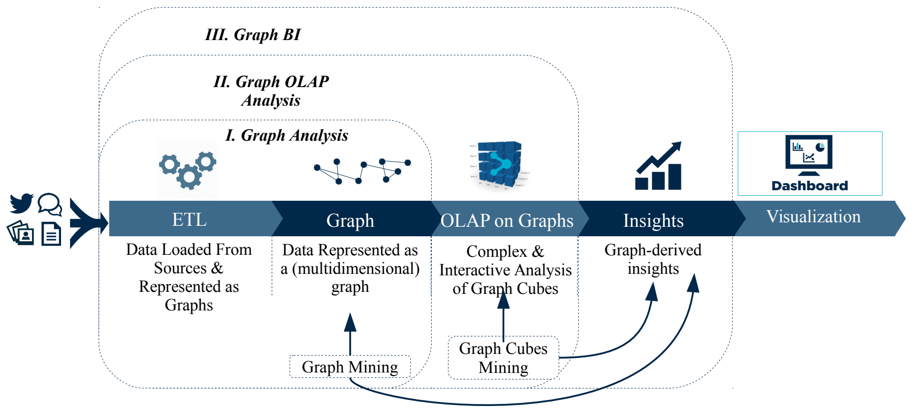


Fig. 6.1: Integration of Graph and Traditional BI

2 Graph Data Modeling

The variety of data structures urges the need for equipping analysts with modeling and querying tools that are aware of the specific nature of each data model. The relational model and its implementations have been developed and matured for decades. However, they were pushed to their limits as the one-size-fits-all data management solutions. The wide adoption of the emerging NoSQL solutions by industry, while not yet as mature as the relational model, proved the need to push databases into fields beyond the traditional business applications. More specifically, organizations experience an urgent need for models and techniques for efficient management of graph data. Indeed, graph models can deal with semantic relativism and heterogeneities, offer the flexibility to combine graphs and support the capability to associate data and metadata.

2.1 Graph Models

According to the literature, the two main families of graphs are property graphs and knowledge graphs:

Property Graphs : Property graphs describe a directed, labeled, and attributed multi-graph. Each real-world entity is represented by a node that contains its label and properties. The label denotes the "type" of the node (i.e., the class to which it belongs). Relationships between entities are represented using edges. The flexibility of property graph models allows the representation of rich structural properties, such as hierarchies and assertions. Property graphs were introduced in the database community to store schemaless data (due to their flexibility to absorb any semantics and attach data with metadata). In the literature, multiple query languages were designed to enable graph-oriented querying of property graphs [9]. However, there is no standard query language for property graphs. Therefore, graph database vendors defined their graph traversal and query languages, such as Cypher and Gremlin. Cypher is an SQL-like declarative language, that uses isomorphism-based no-repeated-edges bag semantics. It was introduced by Neo4j and is centered around pattern matching enriched by built-in algorithmic operators. Gremlin is a graph traversal language, built using Groovy, introduced by Apache TinkerPop3, that uses the homomorphism-based bag semantics.

Knowledge Graphs: The basic formalism behind knowledge graphs, used to describe and link resources, is the Resource Description Framework language (RDF), a W3C recommendation. The basic RDF block is the triple, a binary relationship between a subject and an object; i.e., $< subject, predicate, object >$. The subject and the predicate must be resources (i.e., identified by a URI), whereas the object can be either a resource or a literal (i.e., a constant value such a string or an integer). A set of RDF triples form an RDF graph. RDF Schema (RDFS), a W3C recommendation, was introduced to express basic constraints on RDF triples. In the same line, the Ontology Web Language (OWL) allows expressing richer constraints and semantics. As OWL is serialized on top of RDFS, it results in a graph too. Importantly, it is also usual to refer to knowledge graphs (i.e., RDF(S) or OWL) as ontologies. This is because they can be translated to a fragment of First-Order Logic (FOL), typically, within the family of Description Logics (e.g., in the case of OWL and OWL fragments), and benefit from generic reasoning algorithms in that field. Finally, the W3C recommendation to query knowledge graphs is the SPARQL Protocol and RDF Query Language (SPARQL). Relevantly, SPARQL enables the activation of generic reasoning capabilities when querying knowledge graphs to infer non-explicit knowledge. Knowledge graphs were born within the semantic web stack and therefore initially thought for enabling interoperability and reasoning on semantic data. They are tightly related to the knowledge representation community and therefore thought to represent generic knowledge rather than data as for databases. For this reason, knowledge graph databases are referred to as triplestores rather than graph databases. One key aspect of property graphs is that they provide means (i.e., URI) to universally identify graph vertices and edges from external sources. This facilitates the linking and sharing of data and metadata. However, unlike property graphs and traditional graph databases, which are optimized for graph traversal, they are primarily optimized for handling RDF triples. Another difference is that in property graphs properties could be directly added to edges as well as vertices. In essence, however, knowledge graphs are also graphs and can benefit from the traditional graph algebras presented in the database field.

2.2 Graph Management

Orthogonal to the previous classification there are two main approaches widely used for graph data management (regardless of property or knowledge graphs) at the logical/physical level. The first consists of the use of native graph data models and database engines. The second leverages alternative models, mainly the relational model. For the latter, the graph data is represented by a set of tables, i.e., node tables and edge tables. Traditionally, relational-based graph database engines have been related to triplestores and knowledge graphs, whereas native graph database engines were related to property graphs. Nevertheless, this is nowadays changing and it is currently possible to find native databases for knowledge graphs. These approaches are discussed next:

Native Graph Data Models : In recent years, the trend in developing graph data management systems has shifted to the development of native, relationship-oriented graph databases. Most native graph databases implement the property graph model or a variation of it. The problem of impedance mismatch is resolved since relationships are first-class citizens, and the data is represented as it is perceived without the need to project it on an intermediate representation. The data model is more straightforward to design, and the queries are more intuitive to formulate [61]. From a performance perspective, graph databases are optimized for graph traversals. The cost of traversing an edge is constant, and the overall cost of arbitrary navigation through the graph is much lower than the equivalent joins in relational tables. Subsequent implementation aspects such as graph query processing, indexing, storage, matching, and retrieval which are specifically developed and tuned for graph workloads lead to better performances, especially for queries requiring multiple joins, or containing cycles or other complex patterns. However, they perform worse than the relational-based engines for analytical queries that perform scans over the whole graph. In the software market, established BI vendors are aware of the potential of native graph solutions and have already developed many graph databases such as Neo4j, DataStax Enterprise Graph, Oracle Spatial and Graph, Microsoft GraphEngine, IBM Graph, and Amazon Neptune.

Relational-based Database Models : This approach benefits from the well-established relational model features, and enables smooth integration with a wide range of relational platforms. However, the relational model and its implementations fall short of meeting the requirements for (1) intuitive data modeling, (2) topology-aware graph querying (such as path retrieval and comparison, and graph pattern matching), and (3) traversal-optimized performances. Mapping graph data to relational representation raises the problem of impedance mismatch at the modeling and querying levels. For example, due to the fundamental difference between the two models, the transformation of graph data to the relational model is a manual, complicated process, with a high risk of information loss during the transformation process. The relational model was designed to handle sets of records and transactions instead of entities and connections. The relational query languages and query processing engines are optimized to perform table scans instead of traversals. Graph traversal is often simulated using expensive join operations, which incurs a heavy workload, especially for highly interconnected tables. Moreover, the SQL is not suited to target the topology of the graph with queries such as pattern matching, neighborhood, or path retrieval.

3 Graph Analytics

A plethora of graph analysis techniques was proposed in the literature to reveal interesting properties about the graph topology and the connectivity between graph elements. The core analysis operations of graphs are (1) graph traversal to assess reachability, find shortest paths, and retrieve the neighborhood, (2) metrics computation of local (e.g., centrality), and global properties (e.g., diameter), and (3) graph matching, such as subgraph isomorphism and pattern matching [9]. Most of these operations are supported by graph database engines. This section describes more advanced graph analytics and focuses on three pillars of graph analytics: graph OLAP, graph mining, and graph processing.

3.1 OLAP on Graphs

The multidimensional model is widely used to represent data in the warehouse. The business facts are stored following the multidimensional model in cubes that embed aggregated data denoted as measures, which are the metrics for the analysis. The measures are placed into the so-called multidimensional space, where dimensions are the factors influencing the values of the measures. OLAP techniques are widely used by BI analysts to conduct interactive and complex querying over a large volume of data, from different perspectives and through different hierarchical levels, highlighting the items of interest, and then drilling down to the underlying data from which facts were inferred. The main approaches for the multidimensional design and

OLAP analysis of cubes on graphs are:

Graph OLAP was among the first attempts to design a conceptual framework for OLAP analysis over a collection of homogeneous graphs [28]. Each graph of the collection is considered as a snapshot. Attributes are considered as the dimensions and could be either attached to the whole graph snapshot or attached to a subset of nodes. In the first case, the attributes of the snapshots are called informational dimensions. The aggregations of the graph are performed by overlaying a collection of graph snapshots and merging those with shared informational values. The analysis is referred to as informational OLAP aggregations and consists of edge-centric snapshot overlaying. Thus only edges are merged and changed, with no changes made to the nodes. In the second case, the attributes of the nodes are called topological dimensions. Topological OLAP aggregations consist of merging nodes and edges by navigating through the nodes' hierarchy. Qu et al. introduced a more detailed framework for topological OLAP analysis of graphs [102]. The authors discussed the structural aggregation of the graph following the OLAP paradigm. They presented techniques based on the properties of the graph measures for optimizing measure computations through the different aggregation levels. Berlingerio et al. [19] defined a multidimensional model similar to Graph OLAP, but where the dimensions are the labels of the edges and presented a set of analytical graph-based measures.

Graph Cube is a framework for multidimensional analysis and cube computation over the different levels of aggregations of a graph [143]. It targets single, homogeneous, node-attributed graphs. A subset of the attributes of the nodes is considered as the dimensions. Following these so-called dimensions, the graph cube is obtained by restructuring the initial graph in all possible aggregation. Given n dimensional attributes, the framework introduces the cuboid query, which generates 2^n aggregate graphs (called graph cuboids). Crossboid is a second query introduced by Graph Cube to analyze the interrelationships between different graph cuboids. Pagrol is a MapReduce framework for distributed OLAP analysis of homogeneous attributed graphs [135]. Pagrol introduced the notion of Hyper Graph Cubes that extends the model of Graph Cube by considering the attributes of the edges as dimensions and introduced various optimization techniques for cubes computation and materialization. Ghrab et al. [49] extended those models with a framework for building OLAP cubes on heterogeneous attributed graphs. They presented an extension of property graphs tailored for multidimensional analysis and supporting dimension hierarchies.

Graph OLAP on RDF There is an active research line to generate OLAP cubes on top of RDF and RDF(S) graphs. Nebot [93] and Kämpgen [73] were two of the main attempts to bridge both areas. The former proposes a semi-automatic method for on-demand extraction of semantic data into an MD database. In this way, data could be analyzed using traditional OLAP techniques. The latter studies the extraction of statistical data published using the QB vocabulary, a W3C standard, into an MD database. Both approaches moved the semantic data to a traditional data warehouse. Subsequent attempts avoided such approach and query graph data in an OLAP manner without moving it. For example, Beheshti et al., introduced a distributed framework for OLAP on RDF data [16]. They proposed GOLAP, a graph model for OLAP on graphs, and FSPARQL an extension to SPARQL for OLAP querying of RDF data. GOLAP introduced a rule-based approach for defining new dimensions on the graph. However, it was not until the definition of the QB4OLAP vocabulary that cubes on RDF graphs could not be guaranteed to be MD-compliant. In [131], Varga et al. discuss the drawbacks of previous vocabularies, such as QB, to properly represent MD data and how QB4OLAP overcomes them. This way, resulting cubes can be properly analyzed with traditional OLAP algebras. Relevantly, Pratap Deb Nath et al. present a framework to conduct ETL transformations on top of graph data to produce QB4OLAP-based cubes [92].

3.2 Graph Mining

Data mining refers to the process of discovering patterns or models for data. In contrast to querying that retrieves known patterns, mining enables the discovery of previously unknown, implicit information and knowledge embedded within a dataset. Traditionally, data mining techniques process data as a collection of independent instances (i.e. observations). However, the recent emergence of graph structure as a rich data model involves a paradigm shift on how data mining algorithms can be applied. Graph mining algorithms provide a new way of extracting and discovering latent insight from graphs by leveraging the relationships between entities. However, graph mining algorithms face three main challenges: (1) adapting the mining algorithms to make them graph-aware, (2) redesigning the algorithms to be implemented by those new high-performance techniques, and (3) storing and exploiting multiple but related graphs that serve for the same business purpose as in the graph warehouse.
A plethora of graph mining techniques were proposed in the literature such as graph clustering, frequent subgraph mining, proximity pattern mining, and link prediction. These techniques are relevant in the BI context as they reveal interesting properties about the topology and the connectivity between business entities. For example, consider the case of recommender systems in

e-commerce [79]. The domain could be represented as a bipartite purchase graph, with two types of nodes representing products and customers. An edge is added between a product and a customer if the latter has bought the product. Using graph mining such as graph clustering, two other graphs could be derived: (1) client similarity graph, and (2) product similarity graph. Mining these three graphs enables advanced analysis scenarios such as (1) customer profiling by detecting customer groups using community detection, and the leaders within each group using centrality, (2) product segmentation by detecting products representative of each segment, and (3) using link prediction, targeted marketing personalized to the customer's profile and tailored by current product trends.

The historical and integrated view provided by the data warehouse makes it a suitable backbone for offering a variety of analysis scenarios. In the graph warehouse context, graph mining could be combined with OLAP to offer more capabilities both during the phase of the design and also the analysis of the graph warehouse data. During the design phase, graph mining algorithms could be used to enrich the OLAP cubes with new types of topological dimensions and measures (e.g., PageRank, community). During the analysis, graph mining could assist the analyst in complex tasks such as building summarized business-oriented views of the graph, providing new perspectives to analyze the graph, or discovering interesting or anomalous patterns within the large graph cube space. In the context of outlier detection, graphs provide an elegant framework to predict and describe outliers. For example, in the context of graph cubes mining, [41] developed a measure of interestingness of patterns in a graph cube, while [21] proposed an entropy-based filter to detect interesting associations between attributed nodes in a graph cube.

3.3 Graph Processing

To deal with large and evolving graphs, which is the case in data warehouses, graph BI systems need to integrate large-scale graph processing frameworks. Graph processing frameworks are designed to natively support the graph topology, and they offer graph programming models and abstractions to easily implement a multitude of graph algorithms. These frameworks have the capabilities to efficiently perform large scale, ad-hoc, and distributed computations over large graph data that exceed a single machine capacity. They offer features such as automatic graph partitioning, load balancing, network and disk transfer optimization, and fail-over of the processing tasks.
However, distributed graph processing poses additional challenges to centralized or traditional parallel data processing in that [89]: (1) graph structure is irregular which poses challenges to the graph data partitioning and limits parallelism, (2) computation is driven by the structure, which causes a poor memory locality and poses data transfer issues, and (3) algorithms

traverse the partitioned graph in an exploratory way and are iterative by nature, which is I/O intensive. To tackle these challenges and enable efficient large-scale graph analytics, different processing paradigms were introduced [14]:

- Hadoop Family frameworks: MapReduce denotes a programming model for large-scale data processing. Hadoop is an open-source framework that supports data-intensive distributed applications and clones Google's MapReduce framework. It is designed to process a large amount of unstructured and complex data and runs on shared-nothing architectures. MapReduce frameworks are useful for content-based aggregation of graphs (e.g., graph cube aggregation), but they are not efficient for graph-specific computations[42].

- Synchronous frameworks: Pregel [90], and its open-source implementation Apache Giraph, are distributed fault-tolerant graph processing frameworks designed to execute vertex-centric graph algorithms following the Bulk Synchronous Parallel processing (BSP) paradigm. BSP is a shared-nothing processing paradigm for parallel algorithms execution. The computation is done as a series of super-steps over a set of processing units, each having its local memory. Each super-step consists of three phases, first (1) each processing unit performs concurrently and locally its computations, then (2) data is exchanged between the different processes, finally (3) when a process finishes the computation and communication, it reaches the synchronization barrier and it waits for the rest of processes to finish before proceeding to the next super-step. The advantage of this paradigm is that it ensures a deadlock-free computation. However, the downside is the execution time, where the system has to wait for the slowest machines to finish before proceeding.

- Asynchronous frameworks: In contrast to the synchronous shared-nothing processing frameworks, GraphLab [87] and PowerGraph [54] are asynchronous and follows the Gather-Apply-Scatter computational model, with shared memory abstraction. These frameworks might provide better performances, but incur more complexity and higher scheduling and consistency costs.

- Hybrid Systems: These frameworks enable a mixed workload of graph-parallel and data-parallel processing. GraphX [55] is a component of Apache Spark [141] developed for graph processing . It is a fault-tolerant, distributed, in-memory graph processing framework built on top of the Resilient Distributed Dataset abstraction. GraphX provides a set of primitive operators to load and interactively query the graph data. GRADOOP is a distributed framework for graph management and querying [70]. It introduces a new graph model that extends property graphs, supports Cypher queries, and the queries are processed using Apache Flink [26].

4 Future Research Directions

This paper calls for the development of novel intelligent, efficient, and industry-grade graph warehousing systems. The potential directions include (1) further research on solving complex graph problems (e.g., subgraph isomorphism and graph partitioning), (2) building native graph components (e.g., native graph ETL operations, graph OLAP engines, and a multidimensional query language for graphs), and (3) the integration of artificial intelligence techniques to enable self-service BI for business users. To this end, machine learning should be integrated within the BI systems to automate the warehousing tasks from data preparation, to complex modeling and augmented analytics of graphs (e.g., automated discovery of multidimensional concepts, detection of interesting patterns, and forecasting of business KPIs evolution).

The modules missing for developing an industry-grade graph BI and analytics system are unified in the envisioned architecture presented in Figure 6.2, and they summarize the future research directions as follows:

- Graph Extraction (1): This module allows the extraction of graph data from different data sources that could initially be in various formats and flowing at various rates such as graph streams. The data is cleaned to only capture entities that satisfy the quality constraints (e.g., contains the required attributes with valid values), which guarantees the reliability of data.

- Graph Construction & Enrichment (2): The captured graph data is integrated and formatted according to a given graph model. A promising research direction is the development of graph-aware ETL processes with native graph manipulation operations augmented with machine learning capabilities. ML techniques such as Information Retrieval and Automatic Natural Language Processing could help in the automated extraction and construction of multidimensional graphs from unstructured data such as text. For example, geo-location and sentiment analysis could be applied to enrich the attributes of the data entities and therefore equip businesses with the capability to analyze data from new perspectives. New graph entities could be discovered as well. For instance, using community detection, new labels could be added to the nodes, and using similarity, new edges could be added between the similar nodes. Multiple variations of the traditional ETL approach exist

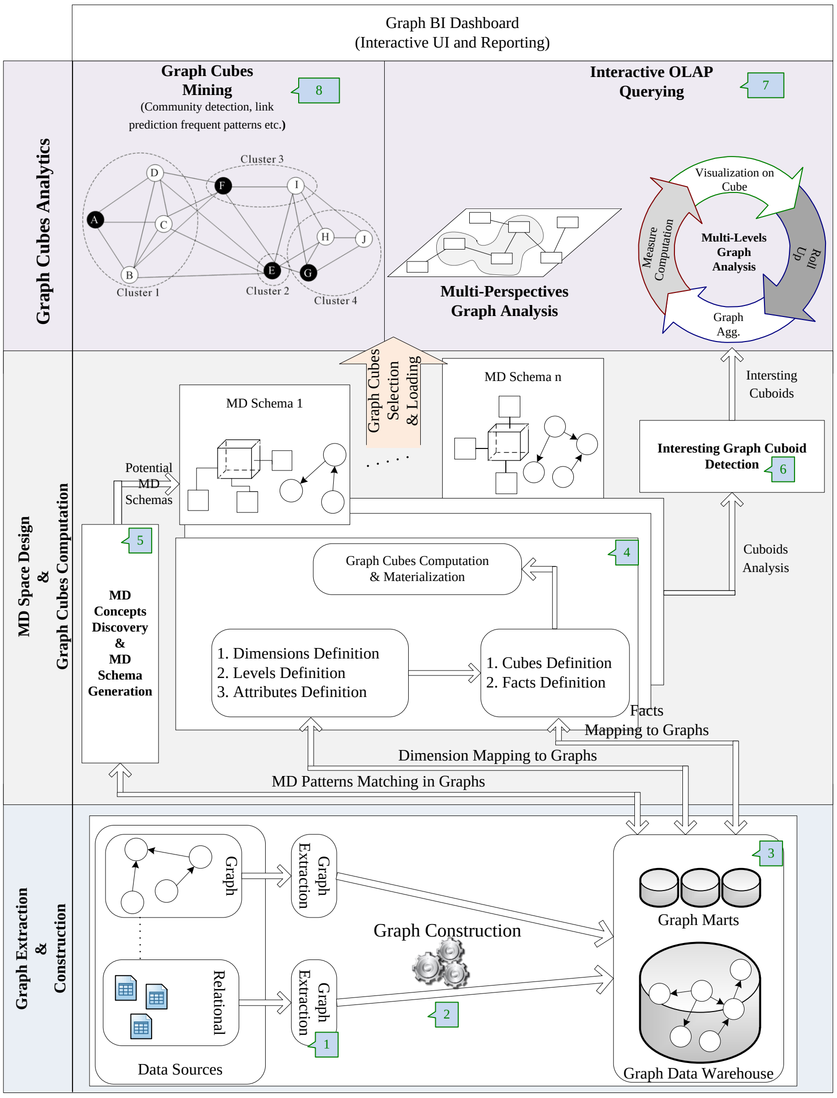


Fig. 6.2: Architecture of a Graph BI & analytics framework

in the literature and might be worth exploring for Graph ETL such as the Extract-Load-Transform, or the Capture-Transform-Flow.

- Graph Data Warehouse (3): The graph data warehouse is the reference central information repository for graph-based decision making. Data is extracted from different sources and integrated using a common graph model. The cleansed and integrated data is natively stored and managed as a multidimensional graph in the graph warehouse. Whereas that data would be transformed into tables in traditional data warehouses. Nevertheless, the conceptual layer remains the same (i.e., represented as dimensions and facts). The changes are related to the logical and physical levels. The graph warehouse provides a suitable backbone for natively analyzing graphs with BI tools such as graph OLAP and graph mining.

- Cube Design and Computation (4): The semantic relativism inherent in graphs allows us to create several views from the same data and make them co-exist in a much simpler way than other data models. Afterward, given a graph lattice, the graph cubes enable the computation and the aggregation of corresponding graph cuboids. Once the required graph cuboids are computed, the result is persisted in the corresponding data marts. To leverage graph properties, graph cubes embed graph-structured measures and dimensions. There is a need for cube computation and aggregation libraries capable of efficiently handling graphs. This line of research includes optimizations such as graph cuboids materialization, indexing, and graph icebergs.

- Discovery of multidimensional concepts and definition of potential multidimensional schemas (5): Multiple multidimensional schemas could be built from the same graph warehouse to satisfy the various analysis needs. Real-world graphs, such as social networks, are complex, dynamic, and flexible. Interesting graph entities might be hidden in the large data sources. Therefore, there is a need for novel graph-aware approaches that enables automatic detection and extraction of multidimensional concepts from large complex graphs. This could be done through the analysis of the topological aspects of graphs, and the projection of the properties of the multidimensional models on them. This will help end-users cope with the complexity and large volume of graphs, and expose potential interesting discovery to decision-makers.

- Assistance with the analysis and synthesis of graphs (6): Given the complexity and large size of the initial graph, there is a need for intelligent modules capable of performing an automated preliminary analysis of the graph to guide the analyst during the exploration of the graph cubes. The goal is to enable self-service BI and facilitate complex tasks such as the extraction of meaningful graph summaries, the discovery of interesting phenomena in the graph cuboids such as frequent graph patterns, and outlier relationships.

- Mining and querying OLAP cubes (7-8): Complex and interactive OLAP analysis and mining of graph cubes are performed at this phase. In contrast to traditional OLAP, graph cubes enable the multidimensional analysis of graph metrics stored in the graph cuboids. For example, analysts could examine the centrality of leaders from multiple perspectives, or identify the communities and their connections at different levels of aggregations. To this end, there is a need to develop graph OLAP engines that support graph-structured cubes. Besides, the Online Analytical Mining of graph data is a promising research direction to empower graph OLAP with mining capabilities. Graphs are dynamic and enabling OLAP on evolving networks by analyzing changing facts and dimensions will help in understanding the structural and informational evolution of networks. Many BI vendors have already integrated graphs into their BI solutions. However, the support for graphs is still limited and there is still a need to push further the integration of graph-derived insights into the decision-making process.

5 Conclusion

Graphs are interesting structures that provide a solid foundation for intuitively representing various domains and solving complex problems while enabling better performance. Graph analytics leverage structural and content-based information to create added-value services, and extend current solutions with new topology-enabled capabilities. In this paper, we surveyed the state of the art on graph BI and analytics and proposed an architecture of Graph BI and Analytics platform augmented with machine learning capabilities, which lays the foundations for promising future research directions. In all, graph analytics has a bright future, and this paper calls for more attention from academia and industry to build next-generation graph-powered BI and analytics frameworks.

Chapter 7

Conclusion and Future Work

In this chapter, we summarize the main results of this Ph.D. thesis and propose promising research directions towards building industry-grade graph data warehouses.

1 Conclusions

Graph warehousing is an emerging topic that extends current decision-making systems with graph management and analysis capabilities. Graphs are indeed a generic structure with a wealth of scientific literature, a strong mathematical foundation, and various real-world applications. A multitude of domains (e.g., transportation, telco, healthcare, etc.) are naturally represented as graphs, and many modern applications benefit from graph analytics (e.g., shortest paths, churn prediction, drug discovery, etc.). Therefore, the topic of graph warehousing received a lot of interest from academia and industry in recent years [103].
In this thesis, we presented our contribution as an end-to-end graph warehousing framework. This work was conducted in three major directions. First, we designed GRAD, a graph database model for analytics and particularly for warehousing. Second, we studied the projection of the conceptual multidimensional model on graphs, extracted novel cubes from graphs, and discussed OLAP analysis of these cubes. Third, we proposed an architecture of the graph data warehouse and described its main building blocks and the remaining gaps. The various components of our graph warehousing framework can be effectively leveraged as a foundation for designing and building industry-grade graph data warehouses, and can ultimately help achieve a deeper understanding and more efficient manipulation of large and complex graphs common in real-world applications.

The main body of this thesis is composed of four papers, included in

Chapters 3 to 5. We review in the remainder of this section the main contributions of each chapter:

- In Chapter 3, we studied the topic of **graph database modeling**. A database model defines the data structures, integrity constraints, and manipulation operators necessary for representing, querying, and guaranteeing the consistency of data. Thereby, it provides the theoretical foundation for building graph databases, which in turn are the central component for the storage and querying of graph data. Given its importance, this topic received a lot of attention from the database community [10]. Many graph database models were proposed in the literature, each adopting a custom design choice such as attributes and labels of graph elements or directions of edges, etc. However, most of the previous work was designed to handle OLTP workloads, thereby not being tuned for analytical processing and particularly warehousing and OLAP workloads. To address this issue, we started from a thorough analysis of the current state-of-the-art to identify potential drawbacks and formulated the first open question tackled in this thesis as **" how to model a graph to make it ready for warehousing and OLAP analytics?"**. That is, what are the graph data structures adapted for graph warehousing, what are the algebraic operators required for the analysis of these graph structures, and what are the integrity constraints that need to be enforced to guarantee the integrity of the graph database?
  We answered these questions by designing **GRAD** (GRAph Database model), a complete graph database model tailored for warehousing and OLAP analytics. Indeed, even though graphs are flexible and most graph models do not enforce a strict schema, an analytical tool needs to be aware of and ensure the mapping between the graph structures to their analytical counterparts. GRAD provides means to express multidimensional concepts and constructs that facilitate the later analysis. It provides (1) analytical graph structures that capture rich semantics, such as encapsulation and hierarchies, (2) a set of operators to manipulate the graph and extract its patterns, and (3) a set of rules to enforce and preserve the integrity of the graph data through the different integration and transformation operations. Nodes and edges are first-class citizens that are self-descriptive and semantically rich as they capture traditional modeling concepts and project them on graphs. Besides, we defined two families of integrity constraints. The first guarantees the graph entity integrity, by forcing each real-world entity to be uniquely represented and identified. The second is semantic constraints, which are domain-specific and represent the user-defined assertions on the graph elements enforced by graph pattern checking. The algebraic operators enable the traversal and pattern matching of GRAD structures

and were specifically designed for querying the graph topology while preserving the integrity constraints. GRAD structures are the operands and the return type of all these algebraic operators. We have illustrated, through a set of queries, how GRAD could be used effectively to support analytical scenarios on graphs.

- In Chapter 4, we presented our second framework towards bringing the value of graph analytics to decision-support systems. Indeed, a major step to extend current data warehouses with graph capabilities is to address the need for native multidimensional graph models and re-think OLAP analysis to capture and expose graph topology and content. This objective led to the second research question we answered in this thesis: **"how to design a multidimensional model aware of the topological characteristic of the graph, and how to extract OLAP cubes from it ?"** We answered this second question by designing a **multidimensional graph model**, and showing how to derive OLAP cubes and extend them using GRAD. We defined the multidimensional concepts for heterogeneous attributed graphs and presented the new dimensions and measures that could be computed from graphs. We first projected these concepts on property graphs to find the candidate multidimensional concepts and to build new types of graph cubes such as topological cubes. Then, we extended the proposed multidimensional model using the implicit semantics embedded in GRAD. For example, we used the hypernode structure to extract intra-class dimensions and the entity edges to support dimension hierarchies. In particular, we illustrated how this extension enables supporting advanced concepts such as dimension hierarchies within multidimensional graphs. As a result, we have shown how a graph database model such as GRAD eases multidimensional modeling and discovery of candidate cubes.

- In Chapter 5, we moved further towards building an end-to-end graph warehousing framework. A major issue of the first papers on graph OLAP is that they simply mapped relational-like multidimensional concepts to graphs, which lead to limited exploitation of the graph topology. The graph structure was not well exposed within graph cubes, which limited their readiness for industrial deployment. This issue lead us to the third question tackled in this thesis: **"how to design a multidimensional model aware of the topological characteristic of the graph, and how to leverage these properties in OLAP analysis?"**
  The answer to this question completed the answer to the second question of the thesis. We extended our previous work on graph warehousing by designing **TopoGraph**, a multidimensional graph model that leverages the content and the topology of the graph and supports new types of cubes and queries combining graph-oriented and OLAP

querying. TopoGraph goes beyond traditional OLAP cubes by considering the topological properties of the graph elements. And it goes beyond current graph warehousing models by proposing new types of graph-structured cubes. These cubes embed a rich repertoire of measures and dimensions that could be represented with numerical values, with entire graphs, or as a combination of them. We studied the correspondence between graph cubes and traditional OLAP cubes and concluded that current warehousing systems are not designed to support graph cubes, which motivates the need for native graph warehousing systems. Indeed, the assumption that graph cuboids could be loaded into relational OLAP cubes might hold for content-based graph cubes. However, loading a graph cuboid into a relational cube causes the loss of the graph structure, which makes relational systems inadequate for topological and graph-structured cubes. Given the graph cubes, we proposed a formal specification of the algebraic operators and discussed the families of queries that they are designed to answer. To validate our proposal, TopoGraph was implemented and experimentally validated with different types of real-world datasets, and used at the core of a social network analysis framework.

- In Chapter 6, we tackled the topic of graph warehousing from an architecture perspective. The variety, complexity, and sheer volume of graphs in a data warehousing context pose challenges to traditional graph storage and analysis tools. Besides, graph cubes have a different structure from their relational counterpart, and OLAP operations exhibit different access and manipulation patterns. Given these specificities of graph warehousing, we identified a need for a unified graph warehousing architecture, designed in the lines of traditional warehousing, but optimized for graph storage and processing, and leveraging topological insights. Therefore, the fourth question we studied in this thesis is **"what are the main building blocks of a graph warehouse, and how could they be unified in a comprehensive architecture?"**
  We answered this question by proposing an architecture of a graph warehousing framework augmented with machine learning capabilities. While adopting a similar template as the traditional BI and warehousing systems, it extends them with graph-aware components that deliver graph-derived insights. The architecture illustrates how the graph data flows from the sources, through the ETL process, the graph warehouse, marts, and graph cubes to the analysis layer. At the core of the proposed architecture, we positioned GRAD as the recommended database model and TopoGraph as the warehousing framework covering graph cubes' computation and analysis. Besides, we enriched the architecture with a set of other complementary components for pro-

cessing optimization and user-assistance purposes. They include the automated discovery of potential multidimensional schema and cubes and the extraction of interesting patterns. This integration of graph warehousing and machine learning lays the foundations for promising future research directions. Moreover, this architecture captures the complete life-cycle of graph data warehousing, and thereby, could be considered as a roadmap, for both academia and industry, towards building next-generation graph BI and analytics systems.

2 Results of the Collaboration with EURANOVA

During the course of this thesis, we had the opportunity to collaborate with our industrial and academic partners to complete our research and explore related topics, with a focus on the efficient processing of graphs and machine learning integration in graph cube analytics. We applied the result of our research to bring innovative solutions to many real-world industrial problems such as fraud detection, product recommendation, and urban planning. We built a multitude of graph libraries and dashboards related to the proposed architecture and integrated the main results of this thesis as modules inside multiple industrial applications.
As for the complementary scientific contributions, they were mostly made in collaboration with master thesis students during their master thesis projects at EURA NOVA, under the supervision of the Ph.D. candidate. In what follows, we describe the published works. The distributed graph cube framework [42], was designed in collaboration with the master student Benoit Denis. The time complexity of Graph Cube algorithms depends on the size of the analyzed network. The centralized approach of the first Graph Cube papers presents major weaknesses for large inputs because (1) the input graph is read sequentially, linearly increasing the execution time of algorithms with its size, and (2) Graph Cube algorithms rely on a centralized in-memory store. A distributed approach providing horizontal scalability was required to handle the Graph OLAP analysis workload. This work provided the first distributed algorithm for graph cubes computation and aggregation using the Map-Reduce model. It was implemented on top of Spark to provide a scalable, efficient, and fault-tolerant solution. Afterward, we designed and implemented GraphOpt [126], with the help of Muaz Twaty. Distributed processing frameworks abstract the distribution of the algorithms and hide complex tasks such as graph partitioning and machine fail-over. However, they each have a large set of specific parameters that require expert knowledge to fine-tune them before efficiently performing large graph jobs. Some frameworks have more than 180 parameters to set, such as in GraphX, which makes the complete search, in the exponentially growing search space, in-

feasible. GraphOpt addresses this issue by proposing an efficient and scalable black-box optimization framework that automatically tunes distributed graph processing frameworks. GraphOpt implemented state-of-the-art optimization algorithms and introduced a new hill-climbing-based search algorithm. These algorithms were used to optimize the performance of two major graph processing frameworks: Giraph and GraphX. Besides, we worked with Florian Demesmaeker to design an algorithm for the discovery of interesting patterns in large graph cubes [41]. Given that a graph cube is composed of a large set of cuboids, another challenge is to discover and extract interesting patterns. The algorithms proposed in the state of the art allow the discovery of interesting patterns in a single graph mainly by analyzing its structure. However, in the case of graph cubes, we need to consider both the structure of the graph and the internal information present in its nodes and edges' attributes to extract the most surprising patterns. Thus, instead of examining all the possible graph cuboid aggregations manually, the proposed algorithm leads the analyst to the interesting associations or patterns in the multidimensional network.

3 Future Research Directions

The development of industry-grade decision-support systems enriched with graphs is still in its early stages. The support for graphs in BI solutions is still limited and there is still a need to push further the integration of graph-derived insights into the decision-making process. In this thesis, we proposed our contribution to advance the state-of-the-art on graph warehousing by proposing a new database and multidimensional model for graphs. Yet, the thesis also opens several interesting research areas for the development of intelligent and efficient graph warehousing systems. The potential research directions include (1) further work on solving fundamental complex graph problems (e.g., subgraph isomorphism and graph partitioning), (2) designing and building missing graph warehousing components (e.g., graph ETL operations, a multidimensional query language for graphs), and (3) the integration of artificial intelligence (AI) techniques to enable self-service graph BI for business users. Integration of AI within graph BI systems could automate various warehousing tasks from data preparation, to complex modeling and efficient querying, and augment the analytics of graphs with new capabilities such as the automated discovery of multidimensional concepts, detection of interesting patterns, and forecasting of business KPIs evolution.

In the remainder of this section, we identify the missing modules for developing an industry-grade graph BI and analytics system and position them in the envisioned architecture presented in Figure 7.1. These missing modules summarize the proposed future research directions.

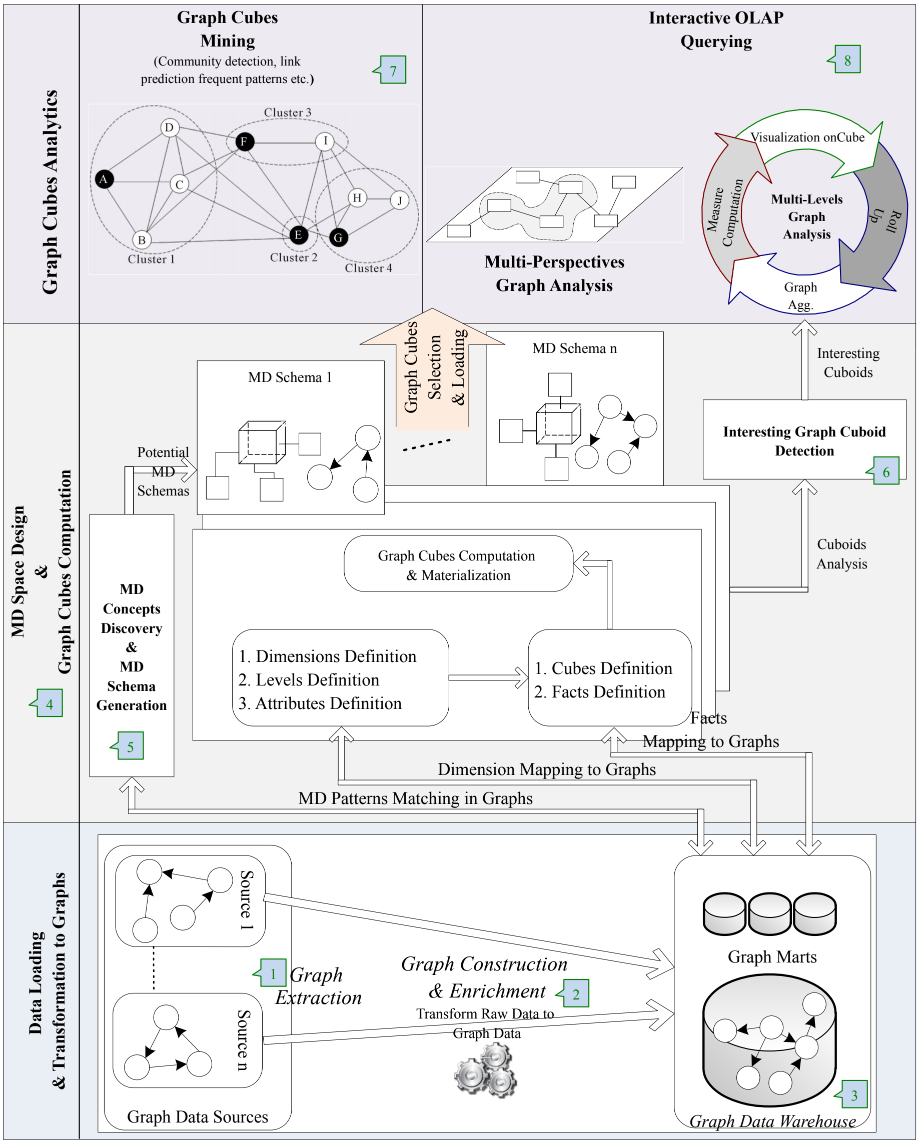

Graph Cubes Analytics
Graph Cubes Mining
(Community detection, link prediction frequent patterns etc.)
7
Cluster 1
Cluster 2
Cluster 3
Cluster 4
Interactive OLAP Querying
8
Multi-Perspectives Graph Analysis
Visualization onCube
Measure Computation
Multi-Levels Graph Analysis
Roll Up
Graph Agg.
MD Space Design & Graph Cubes Computation
4
Graph Cubes Selection & Loading
MD Schema 1
MD Schema n
Potential MD Schemas
Interesting Cuboids
Interesting Graph Cuboid Detection
6
Cuboids Analysis
MD Concepts Discovery & MD Schema Generation
5
Graph Cubes Computation & Materialization
1. Dimensions Definition
2. Levels Definition
3. Attributes Definition
1. Cubes Definition
2. Facts Definition
Facts Mapping to Graphs
Dimension Mapping to Graphs
MD Patterns Matching in Graphs
Data Loading & Transformation to Graphs
Source 1
Source n
Graph Data Sources
1
Graph Extraction
Graph Construction & Enrichment
2
Transform Raw Data to Graph Data
Graph Marts
Graph Data Warehouse
3

Fig. 7.1: Architecture of a Graph BI & analytics framework

- Graph Extraction & Construction (1-2): This module allows the extraction of graph data from different data sources that could initially be in various formats and flowing at various rates such as graph streams. The data is cleaned to only capture entities that satisfy the quality constraints (e.g., contains the required attributes with valid values), which guarantees the reliability of data. The captured graph data is integrated and formatted according to a given graph model. A promising research direction is the development of *graph-aware ETL processes* with native graph manipulation operations augmented with machine learning capabilities. Information Retrieval and Automatic Natural Language Processing algorithms could help in the automated extraction and construction of multidimensional graphs from unstructured data such as text. For example, geo-location and sentiment analysis could be applied to enrich the attributes of the data entities and therefore equip businesses with the capability to analyze data from new perspectives. New graph entities could be discovered as well. For instance, using community detection, new labels could be added to the nodes, and using similarity, new edges could be added between the similar nodes. Multiple variations of the traditional ETL approach exist in the literature and might be worth exploring for Graph ETL such as the Extract-Load-Transform, or the Capture-Transform-Flow.

- Graph Data Warehouse (3): The graph data warehouse is the reference central information repository for graph-based decision-making. The cleansed and integrated data is natively stored and managed as a multidimensional graph in the graph warehouse. In this thesis, we proposed the foundation for the multidimensional modeling of graphs, but many research questions remain open such as supporting *graph data and schema evolution*.

- Graph Cube Design and Computation (4): The semantic relativism inherent in graphs allows creating several views from the same data and make them co-exist in a much simpler way than other data models. Graph cubes are computed from the multidimensional graph and the result is persisted in the corresponding data marts. Given the specific graph properties, such as graph-structured measures and dimensions, there is still a need for *efficient cube computation and aggregation* libraries capable of handling graphs. This line of research includes optimizations such as graph cuboids materialization, indexing, and graph icebergs.

- Discovery of multidimensional concepts and definition of potential multidimensional schemas (5): Multiple multidimensional schemas could be built from the same graph warehouse to satisfy the various analysis needs. Real-world graphs, such as social networks, are complex, dynamic, and flexible. Interesting graph entities might be hidden in the large data sources. Therefore, there is a need for novel graph-aware approaches that enable *automatic detection and extraction of multidimensional concepts* from large complex graphs. This could be done through the analysis of the topological aspects of graphs, and the projection of the properties of the multidimensional models on them. This research direction will help end-users cope with the complexity and large volume of graphs, and expose potential interesting discovery to decision-makers.

- Assistance with the analysis and synthesis of graphs (6): Given the complexity and large size of the initial graph, there is a need for intelligent modules capable of performing an automated preliminary analysis of the graph to guide the analyst during the exploration of the graph cubes. The goal is to enable *self-service graph BI* and facilitate complex tasks such as the extraction of meaningful graph summaries, the discovery of interesting phenomena in the graph cuboids such as frequent graph patterns, and outlier relationships.

- Mining and OLAP querying of graph cubes (7-8): Complex and interactive OLAP analysis and mining of graph cubes are performed at this phase. In contrast to traditional OLAP, graph cubes enable the multidimensional analysis of graph metrics stored in the graph cuboids. To this end, there is a need to develop *graph OLAP engines* that support graph-structured cubes. Besides, *Online Analytical Mining of graph data* is a promising research direction to empower graph OLAP with mining capabilities. Graphs are dynamic and enabling *OLAP on evolving networks* by analyzing changing facts and dimensions will help in understanding the structural and informational evolution of networks.

References

[1] GQL Standard. https://www.gqlstandards.org/home.

[2] A. Abelló, J. Samos, and F. Saltor. YAM2: a multidimensional conceptual model extending UML. *Information Systems*, 31(6):541–567, 2006.

[3] C. C. Aggarwal and H. Wang. Graph Data Management and Mining: A Survey of Algorithms and Applications. In *Managing and Mining Graph Data*, pages 13–68. Springer, 2010.

[4] L. Akoglu, H. Tong, and D. Koutra. Graph-based Anomaly Detection and Description: A Survey. *Data Mining and Knowledge Discovery*, 29(3):626–688, 2015.

[5] E. Andonoff, G. Hubert, A. Parc, and G. Zurfluh. Modelling inheritance, composition and relationship links between objects, object versions and class versions. In J. Iivari, K. Lyytinen, and M. Rossi, editors, *Advanced Information Systems Engineering*, volume 932 of *Lecture Notes in Computer Science*, pages 96–111. Springer Berlin Heidelberg, 1995.

[6] R. Angles. A comparison of current graph database models. In *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, pages 171–177. IEEE, 2012.

[7] R. Angles. The property graph database model. In *Proceedings of the12th Alberto Mendelzon International Workshop on Foundations of Data Management*, 2018.

[8] R. Angles, M. Arenas, P. Barceló, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda, et al. G-core: A core for future graph query languages. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1421–1432, 2018.

[9] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč. Foundations of modern query languages for graph databases. *ACM Computing Surveys (CSUR)*, 50(5):68, 2017.

[10] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, 2008.

[11] M.-A. Aufaure and E. Zimányi. *Business Intelligence: First European Summer School, EBISS 2011, Paris, France, July 3-8, 2011, Tutorial Lectures*, volume 96. Springer Science & Business Media, 2012.

[12] F. Bajaber, R. Elshawi, O. Batarfi, A. Altalhi, A. Barnawi, and S. Sakr. Big data 2.0 processing systems: Taxonomy and open challenges. *Journal of Grid Computing*, 14(3):379–405, 2016.

[13] P. Barceló Baeza. Querying Graph Databases. In *Proceedings of the 32Nd Symposium on Principles of Database Systems*, PODS '13, pages 175–188, New York, NY, USA, 2013. ACM.

[14] O. Batarfi, R. El Shawi, A. G. Fayoumi, R. Nouri, A. Barnawi, S. Sakr, et al. Large scale graph processing systems: survey and an experimental evaluation. *Cluster Computing*, 18(3):1189–1213, 2015.

[15] R. Bean. Variety, Not Volume, Is Driving Big Data Initiatives. https://sloanreview.mit.edu/article/variety-not-volume-is-driving-big-data-initiatives/, 2016. Accessed: 2015-09-25.

[16] S.-M.-R. Beheshti, B. Benatallah, and H. R. Motahari-Nezhad. Scalable Graph-based OLAP Analytics over Process Execution Data. *Distributed and Parallel Databases*, 34(3):379–423, Sep 2016.

[17] B. Benatallah, H. R. Motahari-Nezhad, et al. Scalable graph-based olap analytics over process execution data. *Distributed and Parallel Databases*, pages 1–45, 2015.

[18] C. Berge and E. Minieka. *Graphs and hypergraphs*, volume 7. North-Holland publishing company Amsterdam, 1973.

[19] M. Berlingerio, M. Coscia, F. Giannotti, A. Monreale, and D. Pedreschi. Multidimensional Networks: Foundations of Structural Analysis. *World Wide Web*, 16(5-6):567–593, 2013.

[20] M. Besta, E. Peter, R. Gerstenberger, M. Fischer, M. Podstawski, C. Barthels, G. Alonso, and T. Hoefler. Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries. *arXiv preprint arXiv:1910.09017*, 2019.

[21] D. Bleco and Y. Kotidis. Entropy-based selection of graph cuboids. In *Proceedings of the Fifth International Workshop on Graph Data-Management Experiences and Systems*, GRADES'17, New York, NY, USA, 2017. Association for Computing Machinery.

[22] A. Bonifati, G. Fletcher, H. Voigt, and N. Yakovets. *Querying graphs*. Morgan & Claypool Publishers, 2018.

[23] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. *Computer networks*, 33(1-6):309–320, 2000.

[24] F. Bugiotti, L. Cabibbo, P. Atzeni, and R. Torlone. Database design for NoSQL systems. In *International Conference on Conceptual Modeling*, pages 223–231. Springer, 2014.

[25] I. Cantador, P. Brusilovsky, and T. Kuflik. Second workshop on information heterogeneity and fusion in recommender systems (hetrec2011). In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, page 387–388, New York, NY, USA, 2011. Association for Computing Machinery.

[26] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[27] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[28] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu. Graph OLAP: a multidimensional framework for graph data analysis. *Knowl. Inf. Syst.*, 21(1):41–63, 2009.

[29] H. Chen, R. H. Chiang, and V. C. Storey. Business intelligence and analytics: From big data to big impact. *MIS quarterly*, 36(4), 2012.

[30] M.-S. Chen, J. Han, and P. S. Yu. Data mining: an overview from a database perspective. *IEEE Transactions on Knowledge and data Engineering*, 8(6):866–883, 1996.

[31] P. P.-S. Chen. The Entity-Relationship Model-Toward a Unified View of Data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

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

[33] M. Ciglan, A. Averbuch, and L. Hluchy. Benchmarking traversal operations over graph databases. In *2012 IEEE 28th International Conference on Data Engineering Workshops*, pages 186–189. IEEE, 2012.

[34] E. F. Codd. Data models in database management. In *Workshop on Data Abstraction, Databases and Conceptual Modelling*, pages 112–114, 1980.

[35] D. Colazzo, F. Goasdoué, I. Manolescu, and A. Roatis. Warehousing rdf graphs. In *Bases de Données Avancées*, 2013.

[36] G. Colliat. Olap, relational, and multidimensional database systems. *ACM Sigmod Record*, 25(3):64–69, 1996.

[37] D. J. Cook and L. B. Holder. Graph-based data mining. *IEEE Intelligent Systems and Their Applications*, 15(2):32–41, 2000.

[38] A. Cuzzocrea, L. Bellatreche, and I.-Y. Song. Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions. In *Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP*, pages 67–70. ACM, 2013.

[39] A. Cuzzocrea, D. Saccà, and J. D. Ullman. Big Data: a Research Agenda. In *Proceedings of the 17th International Database Engineering & Applications Symposium*, pages 198–203. ACM, 2013.

[40] K. Dasgupta, R. Singh, B. Viswanathan, D. Chakraborty, S. Mukherjea, A. A. Nanavati, and A. Joshi. Social ties and their relevance to churn in mobile telecom networks. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, EDBT '08, pages 668–677, New York, NY, USA, 2008. ACM.

[41] F. Demesmaeker, A. Ghrab, S. Nijssen, and S. Skhiri. Discovering interesting patterns in large graph cubes. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 3322–3331, 2017.

[42] B. Denis, A. Ghrab, and S. Skhiri. A Distributed Approach for Graph-oriented Multidimensional Analysis. In *2013 IEEE International Conference on Big Data Workshops*, pages 9–16. IEEE, 2013.

[43] L. Duan and L. Da Xu. Business intelligence for enterprise systems: A survey. *IEEE Transactions on Industrial Informatics*, 8(3):679–687, 2012.

[44] W. Fan. Graph Pattern Matching Revised for Social Network Analysis. In *Proceedings of the 15th International Conference on Database Theory*, pages 8–21. ACM, 2012.

[45] D. Feinberg and N. Heudecker. IT Market Clock for Database Management Systems. https://www.gartner.com/doc/2852717/it-market-clock-database-management, 2014. Accessed: 2020-04-02.

[46] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1433–1445, 2018.

[47] M. García-Solaco, F. Saltor, and M. Castellanos. *Semantic Heterogeneity in Multidatabase Systems*, page 129–202. Prentice Hall International (UK) Ltd., GBR, 1995.

[48] A. Ghrab, O. Romero, S. Jouili, and S. Skhiri. Graph BI & Analytics: Current State and Future Challenges. In *International Conference on Big Data Analytics and Knowledge Discovery*, pages 3–18. Springer, 2018.

[49] A. Ghrab, O. Romero, S. Skhiri, A. Vaisman, and E. Zimányi. A Framework for Building OLAP Cubes on Graphs. In *East European Conference on Advances in Databases and Information Systems*, pages 92–105. Springer, 2015.

[50] A. Ghrab, O. Romero, S. Skhiri, A. A. Vaisman, and E. Zimányi. GRAD: on graph database modeling. *CoRR*, abs/1602.00503, 2016.

[51] A. Ghrab, O. Romero, S. Skhiri, and E. Zimányi. TopoGraph: an End-To-End Framework to Build and Analyze Graph Cubes. *Information Systems Frontiers*, pages 1–24, 2020.

[52] A. Ghrab, S. Skhiri, S. Jouili, and E. Zimányi. An analytics-aware conceptual model for evolving graphs. In *Data Warehousing and Knowledge Discovery*, pages 1–12. Springer, 2013.

[53] L. Gómez, B. Kuijpers, and A. Vaisman. Performing olap over graph data: Query language, implementation, and a case study. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics*, pages 1–8. ACM, 2017.

[54] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 17–30, 2012.

[55] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica. Graphx: Graph processing in a distributed dataflow framework. In *OSDI*, volume 14, pages 599–613, 2014.

[56] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data mining and knowledge discovery*, 1(1):29–53, 1997.

[57] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques, Third Edition*, volume 2. Morgan kaufmann, 2011.

[58] L. Hannachi, N. Benblidia, O. Boussaid, and F. Bentayeb. Community cube: a semantic framework for analysing social network data. *International Journal of Metadata, Semantics and Ontologies*, 10(3):155–169, 2015.

[59] H. He and A. K. Singh. Closure-Tree: An Index Structure for Graph Queries. In *Proceedings of the 22Nd International Conference on Data Engineering*, ICDE '06, pages 38–, Washington, DC, USA, 2006. IEEE Computer Society.

[60] H. He and A. K. Singh. Graphs-at-a-time: Query Language and Access Methods for Graph Databases. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 405–418. ACM, 2008.

[61] J. Hölsch, T. Schmidt, and M. Grossniklaus. On the performance of analytical and pattern matching graph queries in neo4j and a relational database. In *Proceedings of the Workshops of the EDBT/ICDT 2017 Joint Conference (EDBT/ICDT 2017), Venice, Italy, March 21-24, 2017.*, 2017.

[62] F. Holzschuher and R. Peinl. Performance of graph query languages: comparison of cypher, gremlin and native access in neo4j. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 195–204, 2013.

[63] F. Holzschuher and R. Peinl. Querying a graph database–language selection and performance considerations. *Journal of Computer and System Sciences*, 82(1):45–68, 2016.

[64] S. Hong, S. Depner, T. Manhardt, J. Van Der Lugt, M. Verstraaten, and H. Chafi. Pgx.d: A fast distributed graph processing engine. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '15, pages 58:1–58:12, New York, NY, USA, 2015. ACM.

[65] W. H. Inmon. *Building the data warehouse*. John wiley & sons, 2005.

[66] C. S. Jensen, T. B. Pedersen, and C. Thomsen. Multidimensional databases and data warehousing. *Synthesis Lectures on Data Management*, 2(1):1–111, 2010.

[67] C. Jiang, F. Coenen, and M. Zito. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(1):75–105, 2013.

[68] X. Jin, J. Han, L. Cao, J. Luo, B. Ding, and C. X. Lin. Visual Cube and On-Line Analytical Processing of Images. In *Proceedings of the 19th ACM International Conference on Information and knowledge management*, pages 849–858. ACM, 2010.

[69] D. Jonker and R. Brath. *Graph Analysis and Visualization: Discovering Business Opportunity in Linked Data*. Wiley, 2015.

[70] M. Junghanns, A. Petermann, K. Gómez, and E. Rahm. Gradoop: Scalable graph data management and analytics with hadoop. *arXiv preprint arXiv:1506.00548*, 2015.

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

[72] E. Junqué de Fortuny, M. Stankova, J. Moeyersoms, B. Minnaert, F. Provost, and D. Martens. Corporate residence fraud detection. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1650–1659. ACM, 2014.

[73] B. Kämpgen and A. Harth. Transforming statistical linked data for use in olap systems. In *Proceedings of the 7th international conference on Semantic systems*, pages 33–40. ACM, 2011.

[74] S. Kang, S. Lee, and J. Kim. Distributed Graph Cube Generation using Spark Framework. *The Journal of Supercomputing*, pages 1–22, 2019.

[75] C. M. Keet and P. R. Fillottrani. Toward an ontology-driven unifying metamodel for uml class diagrams, eer, and orm2. In *Conceptual Modeling*, pages 313–326. Springer, 2013.

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

[77] C. Kiss and M. Bichler. Identification of influencers—measuring influence in customer networks. *Decision Support Systems*, 46(1):233–253, 2008.

[78] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *PVLDB*, 6(2):133–144, 2012.

[79] K. Lee and K. Lee. Escaping your comfort zone: A graph-based recommender system for finding novel recommendations among relevant items. *Expert Systems with Applications*, 42(10):4851–4858, 2015.

[80] H.-J. Lenz and A. Shoshani. Summarizability in OLAP and Statistical Data Bases. In *Proceedings of the Ninth International Conference on Scientific and Statistical Database Management*, pages 132–143. IEEE Computer Society, 1997.

[81] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, june 2014.

[82] C. Li, P. S. Yu, L. Zhao, Y. Xie, and W. Lin. InfoNetOLAPer: Integrating InfoNetWarehouse and InfoNetCube with InfoNetOLAP. *PVLDB*, 4(12):1422–1425, 2011.

[83] E.-P. Lim, H. Chen, and G. Chen. Business intelligence and analytics: Research directions. *ACM Transactions on Management Information Systems (TMIS)*, 3(4):17, 2013.

[84] C. X. Lin, B. Ding, J. Han, F. Zhu, and B. Zhao. Text Cube: Computing IR Measures for Multidimensional Text Database Analysis. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 905–910. IEEE, 2008.

[85] Y. Liu and T. M. Vitolo. Graph data warehouse: Steps to integrating graph databases into the traditional conceptual structure of a data warehouse. In *Big Data (BigData Congress), 2013 IEEE International Congress on*, pages 433–434. IEEE, 2013.

[86] S. Loudcher, W. Jakawat, E.-P. Soriano-Morales, and C. Favre. Combining OLAP and information networks for bibliographic data analysis: a survey. *Scientometrics*, 103:471–487, 2015.

[87] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, Apr. 2012.

[88] H. Luhn. A business intelligence system. *IBM Journal of Research and Development*, 2(4):314–319, 1958.

[89] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry. Challenges in parallel graph processing. *Parallel Processing Letters*, 17(01):5–20, 2007.

[90] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[91] R. R. McCune, T. Weninger, and G. Madey. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Computing Surveys (CSUR)*, 48(2):25, 2015.

[92] R. P. D. Nath, K. Hose, T. B. Pedersen, and O. Romero. Setl: A programmable semantic extract-transform-load framework for semantic data warehouses. *Information Systems*, 2017.

[93] V. Nebot and R. Berlanga. Building data warehouses with semantic web data. *Decision Support Systems*, 52(4):853–868, 2012.

[94] M. Newman. *Networks: an introduction*. Oxford University Press, 2010.

[95] M. E. Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[96] A. Pacaci, A. Zhou, J. Lin, and M. T. Özsu. Do we need specialized graph databases? benchmarking real-time social networking applications. In *Proceedings of the Fifth International Workshop on Graph Data-management Experiences & Systems*, pages 1–7, 2017.

[97] G. A. Pavlopoulos, M. Secrier, C. N. Moschopoulos, T. G. Soldatos, S. Kossida, J. Aerts, R. Schneider, and P. G. Bagos. Using graph theory to analyze biological networks. *BioData mining*, 4(1):10, 2011.

[98] A. Petermann, M. Junghanns, R. Müller, and E. Rahm. Graph-based Data Integration and Business Intelligence with BIIIG. *Proc. VLDB Endow.*, 7(13):1577–1580, 2014.

[99] N. Pobiedina, S. Rümmele, S. Skritek, and H. Werthner. Benchmarking database systems for graph pattern matching. In *International Conference on Database and Expert Systems Applications*, pages 226–241. Springer, 2014.

[100] J. Pokorný. Conceptual and database modelling of graph databases. In *Proceedings of the 20th International Database Engineering & Applications Symposium*, IDEAS '16, page 370–377, New York, NY, USA, 2016. Association for Computing Machinery.

[101] J. Pokornỳ, M. Valenta, and J. Kovačič. Integrity constraints in graph databases. *Procedia Computer Science*, 109:975–981, 2017.

[102] Q. Qu, F. Zhu, X. Yan, J. Han, S. Y. Philip, and H. Li. Efficient topological OLAP on information networks. In *Database Systems for Advanced Applications*, pages 389–403. Springer, 2011.

[103] P. O. Queiroz-Sousa and A. C. Salgado. A Review on OLAP Technologies Applied to Information Networks. *ACM Trans. Knowl. Discov. Data*, 14(1):8:1–8:25, Dec. 2019.

[104] F. Radicchi, S. Fortunato, and A. Vespignani. Citation networks. In *Models of science dynamics*, pages 233–257. Springer, 2012.

[105] R. Raman, O. van Rest, S. Hong, Z. Wu, H. Chafi, and J. Banerjee. Pgx.iso: Parallel and efficient in-memory engine for subgraph isomorphism. In *Proceedings of Workshop on GRAph Data Management Experiences and Systems*, GRADES'14, pages 5:1–5:6, New York, NY, USA, 2014. ACM.

[106] C. Ren, E. Lo, B. Kao, X. Zhu, and R. Cheng. On querying historical evolving graph sequences. *Proceedings of the VLDB Endowment*, 4(11):726–737, 2011.

[107] S. Rizzi. Business intelligence. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems, Second Edition*. Springer, 2018.

[108] M. A. Rodriguez. The gremlin graph traversal machine and language (invited talk). In *Proceedings of the 15th Symposium on Database Programming Languages*, pages 1–10, 2015.

[109] M. A. Rodriguez and P. Neubauer. Constructions from dots and lines. *Bulletin of the American Society for Information Science and Technology*, 36(6):35–41, 2010.

[110] M. A. Rodriguez and P. Neubauer. The graph traversal pattern. In *Graph data management: Techniques and applications*, pages 29–46. IGI Global, 2012.

[111] O. Romero and A. Abelló. On the Need of a Reference Algebra for OLAP. In *Proceedings of the 9th International Conference on Data Warehousing and Knowledge Discovery*, pages 99–110. Springer, 2007.

[112] M. Rudolf, M. Paradies, C. Bornhövd, and W. Lehner. The graph story of the sap hana database. In *BTW*, pages 403–420, 2013.

[113] J. Rumbaugh, I. Jacobson, and G. Booch. *Unified Modeling Language Reference Manual, The*. Pearson Higher Education, 2004.

[114] M. A. Russell. *Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Google+, GitHub, and More*. " O'Reilly Media, Inc.", 2013.

[115] P. J. Sadalage and M. Fowler. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 2012.

[116] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu. The ubiquity of large graphs and surprising challenges of graph processing: extended survey. *The VLDB Journal*, pages 1–24, 2019.

[117] S. Sakr, S. Elnikety, and Y. He. G-sparql: a hybrid engine for querying large attributed graphs. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 335–344. ACM, 2012.

[118] S. Sakr and E. Pardede. *Graph Data Management: Techniques and Applications*. IGI Global, 2012.

[119] C. Shi, Y. Li, J. Zhang, Y. Sun, and S. Y. Philip. A survey of heterogeneous information network analysis. *IEEE Transactions on Knowledge and Data Engineering*, 29(1):17–37, 2017.

[120] S. Skhiri and S. Jouili. Large graph mining: Recent developments, challenges and potential solutions. In M.-A. Aufaure and E. Zimányi, editors, *Business Intelligence*, volume 138 of *Lecture Notes in Business Information Processing*, pages 103–124. Springer Berlin Heidelberg, 2013.

[121] N. Svendsen and S. Wolthusen. Multigraph dependency models for heterogeneous infrastructures. In *International Conference on Critical Infrastructure Protection*, pages 337–350. Springer, 2007.

[122] J. Tang, H. Liu, H. Gao, and A. Das Sarmas. eTrust: understanding trust evolution in an online world. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 253–261. ACM, 2012.

[123] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[124] Y. Tian, R. A. Hankins, and J. M. Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 567–580. ACM, 2008.

[125] Y. Tian, J. M. Patel, V. Nair, S. Martini, and M. Kretzler. Periscope/GQ: a graph querying toolkit. *PVLDB*, 1(2):1404–1407, 2008.

[126] M. Twaty, A. Ghrab, and S. Skhiri. GraphOpt: a Framework for Automatic Parameters Tuning of Graph Processing Frameworks. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 3744–3753. IEEE, 2019.

[127] A. Vaisman and E. Zimányi. *Data Warehouse Systems: Design and Implementation*. Springer, 2014.

[128] W. M. van der Aalst. Process cubes: Slicing, dicing, rolling up and drilling down event data for process mining. In *Asia-Pacific Conference on Business Process Management*, pages 1–22. Springer, 2013.

[129] O. van Rest, S. Hong, J. Kim, X. Meng, and H. Chafi. Pgql: a property graph query language. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, page 7. ACM, 2016.

[130] V. Van Vlasselaer, C. Bravo, O. Caelen, T. Eliassi-Rad, L. Akoglu, M. Snoeck, and B. Baesens. Apate: A novel approach for automated credit card transaction fraud detection using network-based extensions. *Decision Support Systems*, 75:38–48, 2015.

[131] J. Varga, A. A. Vaisman, O. Romero, L. Etcheverry, T. B. Pedersen, and C. Thomsen. Dimensional enrichment of statistical linked open data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 40:22–51, 2016.

[132] P. Vassiliadis and A. Simitsis. Near real time etl. In S. Kozielski and R. Wrembel, editors, *New Trends in Data Warehousing and Data Analysis*, volume 3 of *Annals of Information Systems*, pages 1–31. Springer US, 2009.

[133] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the 48th annual Southeast regional conference*, page 42. ACM, 2010.

[134] P. Wang, B. Wu, and B. Wang. TSMH Graph Cube: A Novel Framework for Large Scale Multi-dimensional Network Analysis. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2015.

[135] Z. Wang, Q. Fan, H. Wang, K.-l. Tan, D. Agrawal, and A. El Abbadi. Pagrol: Parallel graph OLAP over large-scale attributed graphs. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 496–507. IEEE, 2014.

[136] P. T. Wood. Query Languages for Graph Databases. *SIGMOD Rec.*, 41(1):50–60, 2012.

[137] X. Wu, B. Wu, and B. Wang. Pamp;D Graph Cube: Model and Parallel Materialization for Multidimensional Heterogeneous Network. In *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, pages 95–104. IEEE, 2017.

[138] F. Yamaguchi, N. Golde, D. Arp, and K. Rieck. Modeling and discovering vulnerabilities with code property graphs. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 590–604. IEEE, 2014.

[139] S.-R. Yan, X.-L. Zheng, Y. Wang, W. W. Song, and W.-Y. Zhang. A graph-based comprehensive reputation model: Exploiting the social context of opinions to enhance trust in social commerce. *Information Sciences*, 318:51–72, 2015.

[140] M. Yin, B. Wu, and Z. Zeng. HMGraph OLAP: a novel framework for multi-dimensional heterogeneous network analysis. In *Proceedings of the 15th international workshop on Data warehousing and OLAP*, pages 137–144. ACM, 2012.

[141] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[142] P. Zhao, J. Han, and Y. Sun. P-rank: A comprehensive structural similarity measure over information networks. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 553–562. ACM, 2009.

[143] P. Zhao, X. Li, D. Xin, and J. Han. Graph cube: on warehousing and OLAP multidimensional networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 853–864. ACM, 2011.

[144] P. Zhao, J. X. Yu, and P. S. Yu. Graph Indexing: Tree + Delta <= Graph. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 938–949. VLDB Endowment, 2007.

Appendix A

An Analytics-Aware Conceptual Model For Evolving Graphs

The paper has been published in the *Proceedings of the 15th International Conference on Data Warehousing and Knowledge Discovery. (DaWaK 2013).*
The layout of the paper has been revised.
DOI: https://doi.org/10.1007/978-3-642-40131-2_1

Abstract

Graphs are ubiquitous data structures commonly used to represent highly connected data. Many real-world applications, such as social and biological networks, are modeled as graphs. To answer the surge for graph data management, many graph database solutions were developed. These databases are commonly classified as NoSQL graph databases, and they provide better support for graph data management than their relational counterparts. However, each of these databases implements its own operational graph data model, which differs among the products. Further, there is no commonly agreed conceptual model for graph databases.

In this paper, we introduce a novel conceptual model for graph databases. Our model aims to provide analysts with a set of simple, well-defined, and adaptable *conceptual components to perform rich analysis tasks. These components take into account the evolving aspect of the graph. Our model is analytics-oriented, flexible, and incremental, enabling analysis over evolving graph data. The proposed model provides a typing mechanism for the underlying graph, and formally defines the minimal set of data structures and operators needed to analyze the graph.*

1 Introduction

The relational model was considered for several decades as the default choice for data modeling and management applications. However, with the rise of Big Data, relational databases fell short of complex application expectations. Big Data refers to data generated at unpredictable speed, scale, and size from heterogeneous sources, such as web logs and social networks. The distribution and variety of data make ensuring ACID properties, required by the relational model, a very challenging task. This situation has lead to the development of new data models and tools, known as the NoSQL movement. NoSQL models are based on trading consistency for performance according to the CAP theorem, in contrast to relational ACID properties.

NoSQL databases can be divided into four families, namely key/value stores, column stores, document databases, and graph databases. Of particular relevance to this paper is the analysis of graph databases. Graphs have the benefit of revealing valuable insights from both the network structure and the data embedded within the structure [10]. Complex real-world problems, such as intelligent transportation as well as social and biological network analysis, could be abstracted and solved using graphs structures and algorithms. In this paper, we introduce a new graph modeling approach for the effective analysis of evolving graphs. By evolving we mean the variation of the values of an attribute across a discrete domain. Evolution could be over time, quantity, region, etc. In the corresponding non-evolving graph, the information would be discarded when the attributes or the topology changes.

The model introduces a typing system that entails explicit labeling of the graph elements. This might introduce redundancy in the graph since part of the facts could be discovered while traversing the data. However, we tolerate this redundancy in favor of richer and smoother analysis. Trading redundancy for the sake of better performance is a frequent choice when it comes to designing analytics-oriented data stores such as data warehouses. Within the proposed model, we define a set of operators to manipulate analytics-oriented evolving graphs. The goal is to help analysts navigate and extract relevant portions of the graph. Here, we provide a minimal set of operators. Nevertheless, richer analysis scenarios could be achieved by combining these operators.

We consider as a running example of this paper the Epinion product rat-

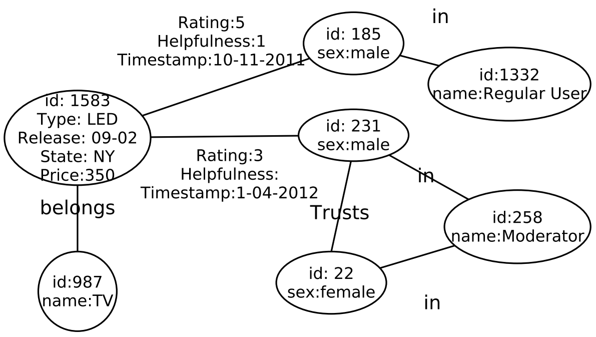


Fig. A.1: Product rating network

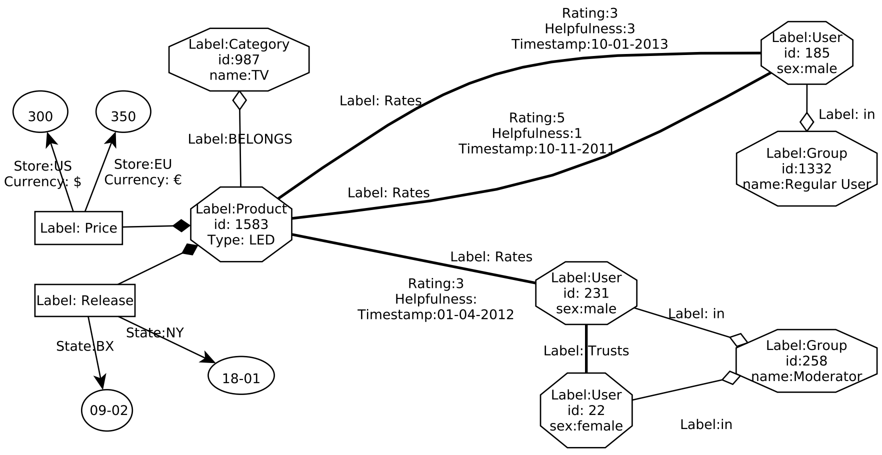


Fig. A.2: Evolving product rating network

ing network [122], shown in Figure A.1. The network is composed of a set of users grouped by group and products grouped by category. Each user has a profile, a list of product ratings, and a linked by trust relationships with other users.

This network is sufficient to answer queries about the average rating of a product, or detection of communities of users. However, information about the evolution of the price by region or the history of the rating of a given product by a user is impossible to obtain. This data is discarded and not versioned for further reuse. Hence, we enrich the original model with a typing system supporting network evolution. The evolving network keeps track of information such as the evolution of the price by region and the history of a product's rating by a user. Figure A.2 depicts a part of the evolving network example. The evolving network could be used to answer rich queries like (1) correlations between sales decrease and product rating evolution, (2) detection of popular and trendy products, and (3) discovery of active and passive users. The previous queries could be then reused in richer scenarios such as (4) recommendation of new products, and (5) targeted advertising for influential people in the network.

The contributions of our work are summarized as follows:

- We define a conceptual model for evolving graphs. The model is designed to handle analytics over large graphs using a novel typing mechanism.
- We propose a comprehensive set of querying operators to perform interactive graph querying through subgraph extraction functionalities.
- We describe a detailed use case of the model by reusing it as the ground for multidimensional analysis of evolving graphs.

The remainder of the paper is organized as follows. In Section 2, we develop a conceptual model to represent analytics-oriented evolving graphs. Section 3 defines the fundamental, general-purpose operators for querying the model. Section 4 demonstrates the usefulness of the proposed model for complex analytics by using it as the basis for multidimensional analysis. Section 5 discusses related work and compares it to our proposed model. Finally, Section 6 sketches future works and concludes the paper.

2 Evolving Graph Model

In this section, we present the evolving graph model serving as the basis for the analysis framework. The **input** to our framework is a directed, attributed, heterogeneous multi-graph. Attributes are a map of key/value pairs attached to nodes and edges of the graph. Nodes (resp., edges) may have different attributes, and multiple edges may link the same pair of nodes.

We first define the typing mechanism that characterizes nodes and edges. We propose three types of nodes, defined next.

Definition A.1. An **entity node** is defined by a tuple $\langle label, K_a, O_a \rangle$ where (1) *label* denotes the type of the entity node, such as user or product, (2) K_a is the map of key attributes that univocally identify the entity node, and (3) O_a is the map of optional attributes. The attributes of an entity node are immutable. The set of entity nodes is denoted as V_{en}.

Definition A.2. An **evolving node** keeps track of the discrete evolution of the attributes of entity nodes. Attributes of entity nodes that are subject to change are typed as evolving nodes. An evolving node contains only a label denoting its name and reflecting the original attribute it represents. Changes are treated as punctual events and reflect the discrete evolution of the attributes. The set of evolving nodes is denoted as V_{ev}.

Definition A.3. A **value node** has a unique attribute representing the value of its corresponding evolving node in a given context. The set of value nodes is denoted as V_v.

We adopt the UML notation for relationships to represent the edges of the graph. With regards to the nodes they link, we classify the edges as follows. Edges linking entity nodes are of two types:

Definition A.4. An **entity edge** (denoted by $\textcircled{V_{en}} \underset{E_{en}}{\text{———}} \textcircled{V_{en}}$) describes the *association* between two entity nodes. The set of entity edges is denoted as E_{en} ($E_{en} \subseteq V_{en} \times V_{en}$).

Definition A.5. A **hierarchical edge** (denoted by $\textcircled{V_{en}} \underset{E_h}{\text{——}\diamond} \textcircled{V_{en}}$) depicts an *aggregation* (i.e., part-of) relationship between two entity nodes. The set of hierarchical edges is denoted as E_h ($E_h \subseteq V_{en} \times V_{en}$).

Both of the above edge types have attributes and labels. If an edge between two entity nodes evolves, it is replicated, and the new one is filled with the new value. We denote an entity (resp. hierarchical) edge as a tuple $\langle label, Atts \rangle$, where $label$ is the type of the relationship, and $Atts$ is the set of its attributes.

Definition A.6. An **evolving edge** (denoted by $\textcircled{V_{ev}} \underset{E_{ev}}{\text{——}\blacklozenge} \textcircled{V_{en}}$) represents a *composition* relationship, i.e. a life-cycle dependency between nodes. It keeps track of the changing attributes extracted as new nodes. The set of evolving edges is denoted as E_{ev} ($E_{ev} \subseteq V_{en} \times V_{ev}$).

Definition A.7. A **versioning edge** (denoted by $\textcircled{V_{ev}} \xrightarrow[E_v]{} \textcircled{V_v}$) denotes a *directed association* between an evolving node and a value node. Evolving edges are attributed, where each attribute is a key/value pair describing the context for the value node. The set of versioning edges is denoted as E_v ($E_v \subseteq V_{ev} \times V_v$).

We introduce now two new data entities oriented for analytics queries.

Definition A.8. An **analytics hypernode** is an induced subgraph[1,2] grouping an entity node, all its evolving and value nodes, and all edges between them. An analytics hypernode whose entity node is v is denoted as $\Gamma_v = (V, E)$, where $V \subseteq (V_{en} \cup V_{ev} \cup V_v)$ and $E \subseteq (E_{ev} \cup E_v)$. Each node (resp., edge) is part of only one hypernode: $\forall u \in V$ (resp., $e \in E$), $\exists! \Gamma_v \mid u \in \Gamma_v$ (resp., $e \in \Gamma_v$).

[1] $G_2 = (V_2, E_2)$ is a subgraph of $G_1 = (V_1, E_1)$ if $V_2 \subseteq V_1$ and $E_2 \subseteq E_1$

[2] G_2 is an induced subgraph of G_1 if all edges between V_2 present in E_1 are in E_2

Definition A.9. A **class** is a label-based grouping. A class denotes a set of analytics hypernodes whose underlying entity nodes share the same label.

With the input graph clearly defined, we introduce the graph model as follows.

Definition A.10. An **analytics-oriented evolving graph** is a single graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \alpha, \beta, \Lambda, \lambda)$, where:

- $\mathcal{V} = \{V_{en}, V_{ev}, V_v\}$ is the set of nodes.
- $\mathcal{E} = \{E_{en}, E_h, E_{ev}, E_v\}$ is the set of edges.
- $\alpha : (V_{en} \cup V_{ev}) \longrightarrow L_V$ is the function that returns the label for each entity or evolving node, where L_V is the set of labels of entity and evolving nodes.
- $\beta : (E_{en} \cup E_h) \longrightarrow L_E$ is the function that returns the label for each entity or hierarchical edge. L_E is the set of labels of entity and hierarchical edges.
- $\Lambda_{key} : (V_{en} \cup V_v) \longrightarrow Dom(value)$ is the function that returns the value of an attribute given its *key*. Λ is applied only to entity and value nodes. $Dom(value)$ denotes the domain of *value*.
- $\lambda_{key} : (E_{en} \cup E_h) \longrightarrow Dom(value)$ is the function that returns the value of an attribute given its *key*. λ is applied only to entity and hierarchical edges. $Dom(value)$ denotes the domain of *value*.

Concerning the Epinion network shown in Figure A.2, *Product* and *User* are entity nodes, while *Price* is an evolving node attached to the products. Users are linked to each other by entity edges labeled *Trusts* and by hierarchical edges to their *Group*. The price keeps track of the evolution of the product price by region. Multiple ratings of the same product by the same user are recorded. The metamodel of an analytics-oriented evolving graph is shown in Figure A.3.

3 Querying the Graph Model

Selection and projection are two fundamental operators in relational algebra, used to extract a subset of data according to predefined conditions on the data tuples. As their names imply, selection selects the set of tuples of a relation according to a condition on the values of their elements. Projection alters the structure of the relation by removing a subset of the elements of the tuples and could be used to add elements by combining existing elements and

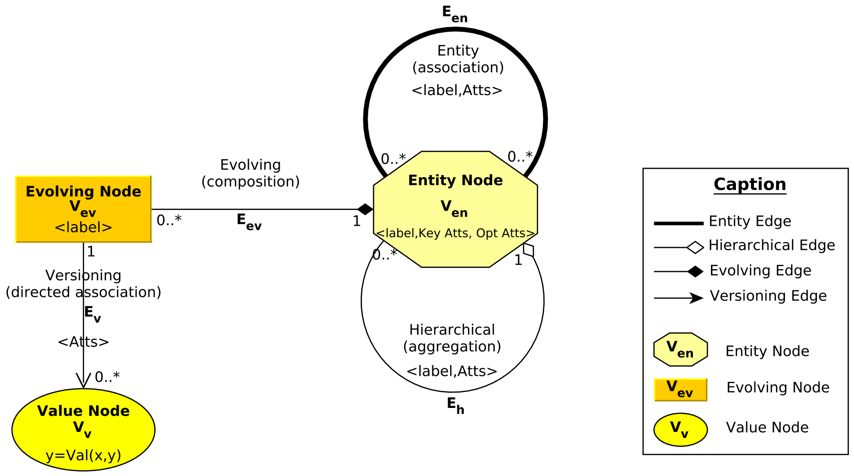


Fig. A.3: Analytics-oriented evolving graph metamodel

constant values. In this paper, we redefine these two operators for evolving graph analysis. Then, we go a step further by introducing the traversal operation that is essential for graph analysis and provides finer control of data extraction. However, we do not cover binary operations such as union and intersection of subgraphs, which we consider out of the scope of the model definition.

All the proposed operators perform subgraph extraction operations. Given an input graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \alpha_1, \beta_1, \Lambda_1, \lambda_1)$, we denote the produced subgraph as $\mathcal{G}' = (\mathcal{V}', \mathcal{E}', \alpha_2, \beta_2, \Lambda_2, \lambda_2)$ where:

- $\mathcal{V}' \subseteq \mathcal{V}$, and $\mathcal{E}' \subseteq \mathcal{E}$
- $\alpha_2(u) = \alpha_1(u), \forall u \in \mathcal{V}'$
- $\beta_2(e) = \beta_1(e), \forall e \in \mathcal{E}'$
- $\Lambda_2(u) = \Lambda_1(u)$, $\forall u \in V'_{en} \cup V'_v$
- $\lambda_2(u) = \lambda_1(u)$, $\forall e \in E'_{en} \cup E'_h$
- $\Gamma_v \subseteq \mathcal{G}'$, iff $v \in V'_{en}$.

These conditions are valid for the three following operators. For the remainder of the paper, asterisk $(*)$ denotes an optional parameter that could be supplied many times and $|S|$ denotes the cardinality of a set S. We start by examining the selection.

Definition A.11. A **selection** $\sigma_{\text{([NLabel,AttVals]*; [ELabel,AttVals]*)}}(\mathcal{G})$ is a *partial*[3] subgraph extraction operation. It is applied on analytics hypernodes

[3] G_2 is a partial subgraph of G_1 if a subset of the edges between V_2 from E_1 is in E_2

and the edges linking their entity nodes. It takes as input a list of the labels ($NLabel$) of the entity nodes V_{en}, underlying the targeted analytics hypernodes, (resp., a list of labels ($ELabel$) of their edges E_{en} and E_h) and the corresponding values of their targeted attributes $AttVals$. A selection returns a partial subgraph $\mathcal{G}'$ of $\mathcal{G}$ where :

- $\alpha_2(u) \in NLabel, \forall u \in V'_{en}$
- $\beta_2(e) \in ELabel, \forall e \in (E'_{en} \cup E'_h)$
- $u \in V'_{en}$ iff $\alpha_1(u) \in NLabel$ and $\exists (k_i, v_i) \in AttVals | \Lambda_{1_{key}}(u) = V_i$, $\forall key = k_i$
- $e \in E'_{en} \cup E'_h$ iff $\beta_1(e) \in ELabel$ and $\exists (k_i, v_i) \in AttVals \mid \lambda_{1_{key}}(e) = v_i$, $\forall key = k_i$
- $u \in \mathcal{V}'$, iff $\exists \Gamma_v \subseteq \mathcal{G}' \mid u \in \Gamma_v$.

In the example of Figure A.2, $\sigma_{\text{(User; Trusts)}}(\mathcal{G}_{epinion})$ detects the communities of users trusting each other. This is accomplished by selecting all analytics hypernodes whose entity nodes are labeled as *User* and linked by the entity edges labeled *Trusts*. The operation presented above is useful for models presenting *intra-class relationships*, i.e. relationships between analytics hypernodes with the same label. A further step is to perform *inter-class selections*. In this case, selection applies on an heterogeneous set of entity nodes and edges. $\sigma_{\text{(Product; User; Rates)}}(\mathcal{G}_{epinion})$ is an inter-class selection. It selects the network comprised of *Rates* relationships, *Product* and *User* analytics hypernodes.

Definition A.12. A **projection** $\pi_{\text{(EvLabel, \{ValSet\})}}\{\mathcal{G}, NLabel\}$ is an *induced* subgraph extraction operation. It is applied on a single class of analytics hypernodes, selected through $NLabel$. Other analytics hypernodes remain untouched by this operation. Evolving nodes whose label is not in $EvLabel$ are removed from the targeted analytics hypernodes. It further narrows the range of values in the resulting subgraph by specifying for each versioning edge a key/value map of the requested values, $\{ValSet\}$. A projection returns an induced subgraph $\mathcal{G}'$ where:

- $\mathcal{E}' = \mathcal{E} \cap (\mathcal{V}' \times \mathcal{V}')$
- $u \in V'_{en}$ iff $\alpha_1(v) \in NLabel$
- $u \in V'_{ev}$ iff :$\alpha_1(u) \in EvLabel$ and $\exists \Gamma_v \subseteq \mathcal{G}' \mid u \in \Gamma_v$
- $u \in V'_v$ iff: $\exists \Gamma_v \subseteq \mathcal{G}' \mid u \in \Gamma_v$ and $\exists\ e = (u, u_e), e \in E'_v \mid u_e \in \Gamma_v$ and $\exists (k_i, v_i) \in ValSet | \lambda_{1_{key}}(e) = v_i, \forall key = k_i$.

For the network of Figure A.2, $\pi_{(\text{Price, (Store,\{EU,ME\})})}(\mathcal{G}_{epinion}, Product)$ acts only on Product hypernodes. It extracts the subgraph containing as evolving nodes only the Price. And for the Price Value nodes, only those representing EU and ME stores are kept, i.e, the US store is dropped from the resulting graph in all Product analytics hypernodes.

Definition A.13. A **traversal** $\mathcal{T}_{(\text{Start,Pattern})}$ is a subgraph extraction operation. A traversal starts from an entity node and the navigation on the graph is guided according to given rules. Traversal only navigates between entity nodes. However, the navigation rules could be applied at any node or edge to decide whether to include the current entity node, i.e., rules are applied to entity nodes attributes as well as any of their analytics hypernode internal edges and nodes. We refer to these navigation rules as patterns, and hence the subgraph extraction becomes a pattern matching operation. A pattern is a finite sequence of conditions dictating the navigation steps. At each node ($u \in V_{en}$) and edge ($e \in E_{en} \cup E_h$), the next step is defined by an expression applied on the labels or the attributes of the current element. For a step i, a pattern dictates the next elements to visit, and could be a combination of the following statements:

- $u \in V_{en} | \alpha(u) = label_i$, dictates the next nodes based on the supplied label
- $e \in (E_{en} \cup E_h) | \beta(e) = label_i$, dictates the next edges based on the label
- $u \in V_{en} | \Lambda_{key}(u) = val_i$, dictates the next nodes based on the supplied attribute value
- $e \in (E_{en} \cup E_h) | \lambda_{key}(e) = val_i$, dictates the next edges based on the supplied attribute value

$\mathcal{T}$ is applied as follows: $\mathcal{T}_{(Start,Pattern)}(\mathcal{G})$ and returns a partial subgraph of the input graph. Steps are separated by dashes $(-)$.

A typical traversal scenario is the following query, applied on the example shown in Figure A.2: *for a user A, return all products she didn't rate, and whose trusted contacts have already rated above four*. Such a query is useful in a recommendation engine. This operation is expressed as follows:
$\mathcal{T}_{(User_A,Pattern)}(\mathcal{G}_{epinion})$, where $Pattern = [e \in E_{en} \mid \beta(e) = Trust] - [*] - [e \in E_{en} \mid \beta(e) = Rates \;\&\; \lambda_{Rating}(e) \geq 4] - [u \in V_{en} \mid (u, User_A) \notin \mathcal{E}]$.

In relational databases, the join operation introduces a heavy workload, especially for highly connected tables [133]. In graphs, data is embedded within nodes connected through edges. The cost of running traversals within graphs is much lower than the equivalent joins in relational tables [115]. This makes graphs more suitable for highly connected data compared to relational tables. Moreover, as explained in Section 5, many of the current graph databases provide partial or full support of ACID properties.

The data structure and operations defined above yield the ground for defining an algebra for evolving graph data. However, this should be further investigated and enriched for the sake of completeness.

4 Multidimensional Graph Analysis

Data warehousing provides a particularly interesting use case for the implementation of our model. The subject-oriented and integrated view of data, provided by the data warehouse, makes it a suitable backbone for common analysis techniques such as reporting, complex querying, and data mining algorithms. We assume that the input graph is designed according to the metamodel defined above. The identification, versioning, and insertion of the incoming nodes and edges in the studied graph are done through the ETL phase. The evolving aspect of the graph brings new challenges to the design of the ETL process. Such issues include, but are not limited to, the definition of new entity nodes and labels, the detection of new evolving attributes, and the attachment of new values to evolving attributes. Due to space limitations, we have chosen to limit the application of our model to OLAP analysis. In this section, we briefly describe multidimensional concepts using the graph model proposed in Section 2.

We limit the study to the structures and a subset of the operators defined in the reference algebra described in [111]. However, further research is needed to device new operators uniquely useful in evolving graphs. OLAP analysis enables us to discover hidden facts and relationships between users and products, such as user satisfaction, the evolution of trendy categories, and influential groups.

Figure A.4 illustrates the proposed multidimensional modeling stack. The physical level switches the focus from elementary nodes and edges to class and inter-class relationships. The logical level encapsulates classes into dimensions and aggregates their relationships. The logical level is similar to ROLAP star schema in that it organizes the studied domain into dimensions and prepares the cube construction. The conceptual level abstracts the graph data using cubes and proposes user-friendly operations. The physical level has been described in detail in Section 2, and serves as the ground for various analysis techniques. We focus now on the logical and conceptual level, relevant to the multidimensional analysis.

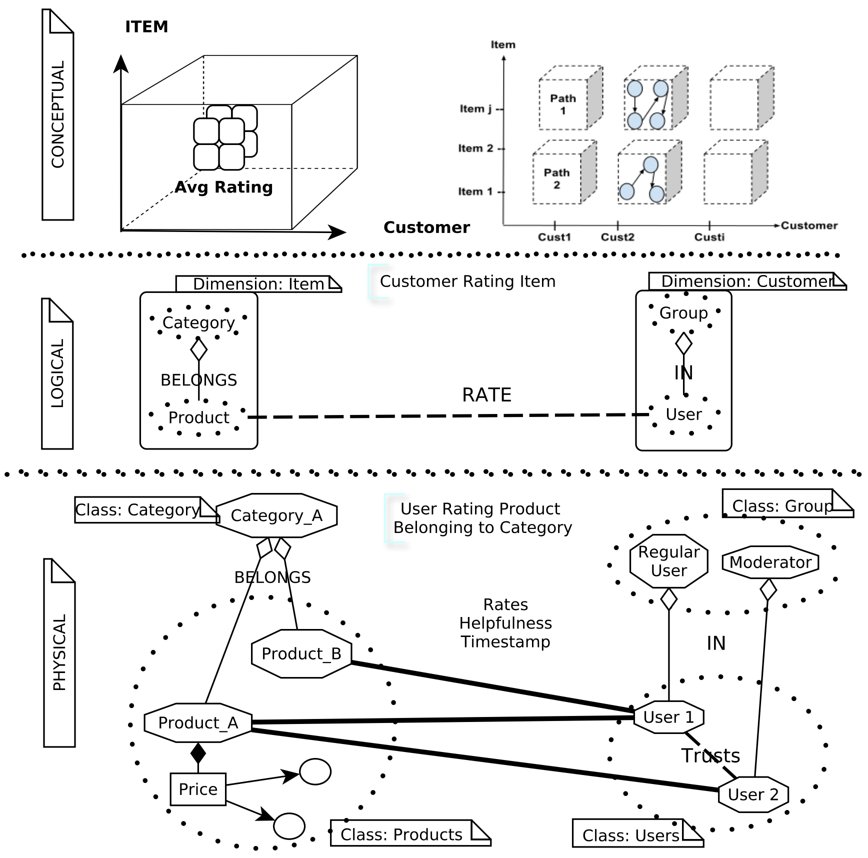


Fig. A.4: Multidimensional analysis of an evolving network

4.1 Data Structures

Dimensions Within our model, a dimension is a tree of classes. A dimension D is defined by a tuple $\langle name, Tree\rangle$, where $name$ denotes the name of the dimension, and $Tree$ is the tree of classes. The root of the tree is the highest class in the dimension hierarchy. A class could be involved in only one dimension. Each level is identified by the class $label$.

In our product rating network example, the $Item$ dimension D_{Item} is denoted as $\langle Item, ILevels\rangle$, where $ILevels = [Product \rightarrow Category]$ and $\rightarrow$ denotes the hierarchical relationship between classes. The shift from class to dimension is depicted in Figure A.4, where the classes Product and Category are grouped in the same dimension, Item.

Measures We distinguish two types of measures, (1) informational measures, calculated from the internal attributes of the edges and nodes such as the average rating of a product, and (2) structural measures, the result of algorithms performed on the structural properties of the graph, such as centrality metrics. For structural measures, a measure could be a subgraph such as the shortest path, or a numerical value such as the length of the path. Using the evolving nature of the graph, we can retrieve further insights such as the evolution of a product rating, or the evolution of the shortest path between users and products. Measures are the metrics used to study a subject. A set of measures showing the data at the same granularity is called a fact. Informational measures are similar to relational measures. Here we focus on the structural measures, specific for graphs.

Cube A cube is a set of cells containing measures and placed in the multidimensional space with regard to a base. A base is the minimal set of dimensions levels that univocally identify a cell within a multidimensional space. A cube $\mathcal{C}$ is defined by a tuple $\langle \mathcal{D}, \mathcal{M}\rangle$, where $\mathcal{D} = \{D_1, D_2, \ldots, D_m\}$ is the set of dimensions, and $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ is the set of measures.

Figure A.4 shows at the left a cube of informational measures (average rating of items by customers) and at the right a cube of structural measures (the shortest path between customers and items).

4.2 Operations

The following representative, but non-exhaustive, set of operations provides a high-level abstraction and is applied at the conceptual level to study OLAP cubes.

Slice Removes a dimension from a cube. $Slice_{[D,Value]}(\mathcal{C})$ operates on the cube $\mathcal{C}$, and returns the subset of cells for which the value of dimension

D is set to $Value$. In the cube of shortest path evolution of Figure A.4, $Slice_{[Item,id=10]}(\mathcal{C})$ limits the set of studied items to one item whose id=10. This cube is computed after extracting the subgraph of the specific item from the graph of all items, through the selection operator of Section 3, $\sigma_{([\text{Product, (id,10)}]; \text{User; Rates})}(\mathcal{G}_{epinion})$

Dice Selects a subset of measures from the cube using a set of conditions on multiple dimensions. This operation is similar to slice but operates on a range of values of a dimension, and on multiple dimensions at the same time. $Dice_{[User,id=10..30;Item,id=1..15]}(\mathcal{C})$, returns a subcube for which users and items identifiers are limited to the specified ranges.

Roll-Up Aggregates classes sharing a common hierarchy using the hierarchical edges. This produces a summary graph with new nodes and edges that are not necessarily present in the original graph. Aggregations could be asynchronous. We could for example study relationships between Category and User rather than Category and Group. The roll-up operators perform structural changes to the graph. If the attributes of the elements involved in the aggregation are additive, an overlay is performed and the values of the attributes are simply incremented. Otherwise, graph summarization techniques such as those discussed on [124, 143, 28] could be used to implement the roll-up operation.

5 Related Work

Graph analytics is gaining a lot of momentum in the data management community in recent years. A survey of graph database models according to their data model, data manipulation operators, and integrity constraints is given in [10]. Current graph databases implement different general-purpose data models, without a commonly agreed conceptual modeling approach. Neo4j[4] is a centralized graph database implementing the property graph[5] model and guaranteeing ACID constraints. Titan[6] is a distributed graph database implementing property graphs and supporting ACID and partial consistency. Graph querying is made either using traversal-oriented languages such as Gremlin[7], SQL-like languages such as Cypher, or through the database core API. Our model could be implemented using any graph database that supports the input graph described in Section 2. RDF is a widespread data model in the Web community and could be an implementation candidate for

[4]http://neo4j.org/

[5]https://github.com/tinkerpop/blueprints/wikiproperty-graph-model

[6]http://thinkaurelius.github.com/titan/

[7]https://github.com/tinkerpop/gremlin/wiki

our conceptual model. Pregel [90], and its open source implementation Giraph[8], are BSP graph processing frameworks designed to execute efficiently graph algorithms. Ren et al. [106] proposed an approach to compute graph-specific measures such as the shortest path and centrality within a graph with gradually changing edge sets. In [76], the authors present a distributed graph database system for storing and retrieving the state of the graph at specific time points. Both of these two papers are based on the redundancy offered by historical graphs trace. The analysis tasks are limited to graph-specific measures and indices with no querying or multidimensional view of data. Moreover, we consider the historical variation as a specific case of graph evolution scenarios. Related research on versioning was done by the database community. In [5], the authors suggested a conceptual model for evolving object-oriented databases by studying the evolution of objects' values, schema, and relationships between the objects. Although some concepts are similar, modeling the versioning depends on the data structures specific for each data model. Multidimensional analysis of graphs data has been first proposed in [28]. The authors introduce informational and topological dimensions. Informational aggregations consist of edge-centric snapshot overlaying and topological aggregations consist of merging nodes and edges by navigating through the nodes' hierarchy. However, the analysis is limited to homogeneous graphs. GraphCube [143] is applied in a single large centralized weighted graph and does not address different edges attributes. Yin et al. [140] introduced a data warehousing model for heterogeneous graphs. They enriched the informational and topological dimensions with the Entity dimension and the Rotate and Stretch operations along with the notion of metapath to extract subgraphs based on edges traversals. However, HMGraph did not discuss the semantics of OLAP operations on the proposed graph data model. Distributed processing frameworks such as Hive [123] propose data warehousing on top of a large volume of data. However, they are considering only the relational model.

6 Conclusions and Future Work

In this paper, we designed a conceptual model for evolving graphs. A plethora of graph database tools are currently developed with multiple management features. However, they do not address the management of evolving networks. Moreover, no common conceptual model for efficient analysis of large evolving networks is agreed upon. We have proposed our contribution to evolving graph analysis by introducing a well defined conceptual model. We illustrated the model with an application on the multidimensional analysis. However, large network analysis requires more work to build a complete

[8] http://giraph.apache.org/

stack of analysis frameworks. As future work, we plan to proceed in warehousing the evolving graphs. Further fundamental operations such as graph aggregations should be investigated for the evolving graphs. A framework for graph data warehousing should integrate an ETL module, which takes care of matching and merging tasks and provides a graph compliant to the proposed model. An exhaustive study of new OLAP operators in evolving graphs is needed. Current graph querying languages such as Cypher should be extended to support multidimensional queries in an MDX-like fashion. Moreover, distributed processing frameworks should be integrated into large graphs processing.

Acknowledgment

This work has been partially funded by the Wallonia Region in Belgium (Grant FIRST-ENTERPRISES N° 6850).