Performing OLAP over Graph Data: Query Language, Implementation, and a Case Study

Leticia Gómez
Instituto Tecnológico de Buenos Aires,Buenos Aires, Argentina
lgomez@itba.edu.ar

Bart Kuijpers
Databases and Theoretical Computer Science Research Group, Hasselt University and Transnational University of Limburg, Belgium
bart.kuijpers@uhasselt.be

Alejandro Vaisman
Instituto Tecnológico de Buenos Aires,Buenos Aires, Argentina
avaisman@itba.edu.ar

ABSTRACT

In current Big Data scenarios, traditional data warehousing and Online Analytical Processing (OLAP) operations on cubes are clearly not sufficient to address the current data analysis requirements. Nevertheless, OLAP operations and models can expand the possibilities of graph analysis beyond the traditional graph-based computation. In spite of this, there is not much work on the problem of taking OLAP analysis to the graph data model. In previous work we proposed a multidimensional (MD) data model for graph analysis, that considers not only the basic graph data, but background information in the form of dimension hierarchies as well. The graphs in our model are node- and edge-labelled directed multi-hypergraphs, called graphoids, defined at several different levels of granularity. In this paper we show how we implemented this proposal over the widely used Neo4J graph database, discuss implementation issues, and present a detailed case study to show how OLAP operations can be used on graphs.

CCS CONCEPTS

• **Information systems** → *Online analytical processing*;

KEYWORDS

OLAP, Graph Databases

1 INTRODUCTION

Online Analytical Processing(OLAP) [14] comprises a set of tools and algorithms that allow querying multidimensional (MD) databases. In these databases, data are modelled as *data cubes*, where each cell contains one or more *measures* of interest, that quantify *facts*. Measure values can be aggregated along *dimensions*, organized as a set of hierarchies. Traditional Online Analytical Processing (OLAP) queries aggregate fact measure data along a set of dimensions, or select a portion of the cube. In Big Data scenarios, graph databases are becoming increasingly popular, although, still, OLAP operations can expand the possibilities of graph analysis beyond the traditional graph-based computation. The present paper addresses this problem.

In previous work, Kuijpers and Vaisman [12] proposed a formal MD data model for graph analysis. Graphs in this model are node- and edge-labelled directed multi-hypergraphs, called *graphoids*, defined at several different levels of granularity, according to dimension hierarchies associated with them. Over this model, graph OLAP operations are defined. These OLAP operations, although analogous to the classic ones, are more powerful and have their own clearly defined semantics. It was also proved that classic OLAP is a particular case of graph OLAP. The running example introduced next, gives the flavour of the hypergraph MD model.

Running example. In Section 5 we present an example that shows the advantages of using graphs instead of the typical relational MD model. For now, we use a simple example to illustrate the MD graph model and its associated operations. The example concerns movies, actors, and movie critics that publish reviews and scores they gave to actors performing in those movies. The base graph is given in Figure 1. The nodes in this example are of the types: #Movie, #Actor, and #Critic, and have an identifier as their first attribute. Further, nodes of type #Movie are described by the movie's name, and the studio which produced the movie. Nodes of type #Actor are described by the actor's name. The hyperedges are of types #Rating, #StarsIn and #DirectedBy. #Rating associates a score and a date with a movie-actor pair. #StarsIn connects actors to movies in which they played. If an actor directed a movie, there is an edge of type #DirectedBy. As background information there are the classic OLAP dimensions, like $Time$, $Movie$, $Actor$, and $Company$, and an $Identifier$ dimension, which is explained in Section 3. Examples of these are shown in Figure 2. Note that this is a simplified example. In a real-world case, for instance, the geographic hierarchy would be shared by other dimensions, or would be itself a dimension, or Producer would be a dimension instead of a level in dimension $Movie$.

Contributions and paper organization. In this paper we: (a) present a proof-of-concept implementation of the data model based on the notion of graphoids; (b) implement a query language based on the graph OLAP operations; (c) discuss a real-world case study, using the Internet Movie Database[1] data. Section 2 discusses related work, while Section 3 reviews the model introduced in [12]. Section 4 presents implementation details. Section 5 discusses our use case, and present the query language, concluding in Section 6.

[1] http://imdb.com

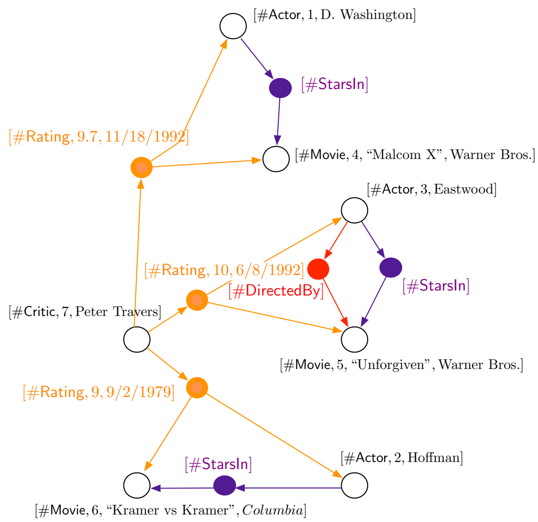

Figure 1: Base movie critic data.

2 RELATED WORK

There is an extensive bibliography on graph database models, comprehensively studied in [1, 2]. Two database models are used in practice: (a) Models based on RDF[2], oriented to the Semantic Web; and (b) Models based on Property Graphs. Models of type (a) represent data as sets of triples of the form $(subject, predicate, object)$, which in turn form an RDF graph. Hartig [7] shows that both models can be reconciled. The present paper is based on the Property Graphs model.

GraphOLAP [3] is a conceptual framework for OLAP on a collection of homogeneous graphs. Aggregations of the graph are performed by overlaying a collection of graph snapshots. Along similar lines, Qu et al. [13] present techniques for topological OLAP analysis of graphs, and propose to optimize measure computation through the different aggregation levels, based on the properties of the graph measures. GraphCube [16] addresses OLAP cubes computation through the different levels of aggregation of a graph, targeting single, homogeneous, node-attributed graphs. Pagrol [15] studies the use of Map-Reduce for distributed OLAP analysis of homogeneous attributed graphs. Also, Distributed Graph Cube [5] is a distributed framework for graph cube computation and aggregation of homogeneous graphs. Finally, in [6] the authors propose a method for defining OLAP cubes from graph data, aimed at extracting the candidate multidimensional spaces in heterogeneous property graphs limited to binary relationships between nodes.

Compared to the works described above, our proposal has a key difference: it supports the notion of OLAP hypergraphs, allowing n-ary, probably duplicated relationships (i.e., multi-hypergraphs), as typically found in real-world Big data scenarios. Some works have addressed hypergraphs in MD databases. For example, in [8] the authors present an approach based on hypergraphs for modelling MD databases for dynamic web-based analysis and adaptive users' requirements. Although they provide some constructs to represent MD elements, OLAP operations are not described, and operations over the hypergraphs are not detailed. This is another important difference between our work and other proposals: we base ourselves on the classic OLAP operations, and formally define their meaning in a graph context. Therefore, a final OLAP user may express queries conceptually, using the operators she knows well, and also take advantage of the graph model flexibility.

[2]https://w3c.org/RDF/

3 PRELIMINARIES AND BACKGROUND

We first review the notions of dimension schema and instance. Details can be found in [10, 11]. Let D be a name for a dimension. A *dimension schema* $\sigma(D)$ *for* D is a lattice, with a unique top, called *All*, and a unique bottom, called *Bottom*, such that all maximal-length paths in the graph go from *Bottom* to *All*. Any path from *Bottom* to *All* in $\sigma(D)$ is called a *hierarchy* of $\sigma(D)$. Each node in a dimension schema is called a *level*. For a dimension schema $\sigma(D)$, and a level ℓ of $\sigma(D)$, a *level instance of* ℓ is a non-empty, finite set $dom(D.\ell)$. If $\ell = All$, $dom(D.All) = \{all\}$. If $\ell = Bottom$, then $dom(D.Bottom) = dom(D)$. A *dimension instance* $I(\sigma(D))$ over $\sigma(D)$ is a directed acyclic graph with node set $\bigcup_\ell dom(D.\ell)$, where the union is taken over all levels in $\sigma(D)$. Further, let ℓ and ℓ' be two levels of $\sigma(D)$, and let $a \in dom(D.\ell)$ and $a' \in dom(D.\ell')$. Then, only if there is a directed edge from ℓ to ℓ' in $\sigma(D)$, there can be a directed edge in $I(\sigma(D))$ from a to a'. If H is a hierarchy in $\sigma(D)$, then the *hierarchy instance* is the subgraph $I_H(\sigma(D))$ of $I(\sigma(D))$ with nodes from $dom(D.\ell)$, for ℓ appearing in H. Also, if a and b are two nodes in a hierarchy instance $I_H(\sigma(D))$, such that (a, b) is in the transitive closure of the edge relation of $I_H(\sigma(D))$, then we say that *a rolls-up* to b and we denote this by $\rho_H(a, b)$. We assume that we work with dimension graphs that guarantee that rolling-up from a through different paths gives the same result [10, 11].

Example 3.1. Figure 2 (left) shows the schema of the background dimensions. Dimension Id in (*e*), represents identifiers (explained later). On the right-hand side, an instance $I(\sigma(Actor))$ for $\sigma(Actor)$ (which is (b) on the left hand side) is shown. □

3.1 The Base graph and Graphoids

To make this paper self-contained, we next present the graph data model in a streamlined fashion (details and proofs are in [12]). We assume that there are dimensions $D_1, \ldots, D_d$ in our application domain, with schemas $\sigma(D_1), \ldots, \sigma(D_d)$, and instances $I(\sigma(D_1)), \ldots, I(\sigma(D_d))$. There is also a special dimension D_0 = Id, called the *Identifier* dimension (Figure 2(e)). As a basic data structure we use the notion of *graphoid* (analogous to a MD cuboid in classical OLAP). A graphoid is composed of attributed nodes and edges. There is a finite, non-empty set $\mathcal{N}$ of *node types*. Nodes are described by *attributes* $\mathcal{A}$, which are levels in the background dimensions, i.e., $\mathcal{A} = \{D.\ell \mid D \in \{D_0, D_1, \ldots, D_d\}$ and ℓ is a level of $D\}$. To each A in $\mathcal{A}$, a *domain* $dom(A)$ is associated. The first attribute in a node type corresponds to the *Identifier* dimension. We assume that a dimension appears only once in a node type. There is also a finite, non-empty set $\mathcal{E}$ of *edge types*, disjoint from $\mathcal{N}$, defined analogously to the node types, except that no identifier dimension is required. Formally, given D_0 = Id, $D_1, \ldots, D_d$, defined as above, and $\ell_1, \ldots, \ell_d$, levels for these dimensions, a $(D_1.\ell_1, \ldots, D_d.\ell_d)$*-graphoid* is a multi-hypergraph (i.e., there can be repeated hyperedges), where all attributes in nodes and edges are defined at the granularity indicated by $D_i.\ell_j$. The $(D_1$.Bottom$, \ldots, D_d$.Bottom$)$-graphoid is called the *base graph*, and is designed to contain all the information of the application domain.

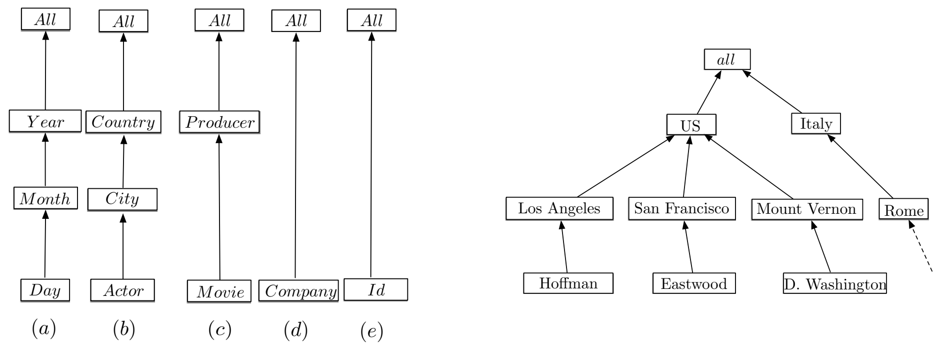

Figure 2: Schemas for the background dimensions in the running example (left); A dimension instance for the dimension *Actor* (right).

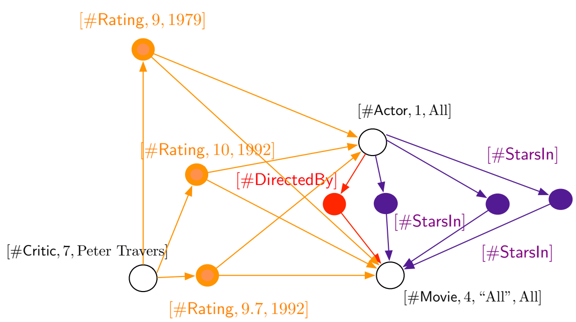

Figure 3: A minimal (Time.Year, Movie.All, Company.All)-graphoid, for data of Figure 1.

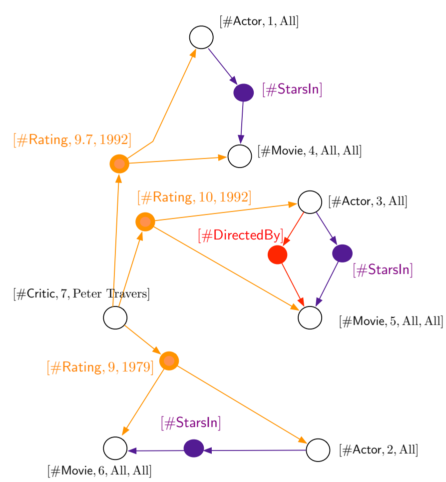

Figure 4: Climbing to the Year level along the *Time* dimension, and to the All level along dimensions *Movie*, *Actor*, and *Company*, for data of Figure 1.

Example 3.2. Figure 1 shows a base graph with node set $N = \{1, 2, 3, 4, 5, 6, 7\}$, and node types #Actor, #Movie, and #Critic. For #Movie, the first attribute is a node identifier, and the other attributes are elements in the bottom levels of dimensions *Movie* and *Company*. The #Rating edge type represents reviews by a critic for an actor-movie pair. The #StarsIn edge type tells who performed in a movie. Finally, #DirectedBy indicates the movie director. □

Note that more than one $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid can exist. The definition of the OLAP operations requires producing a normalized equivalent graphoid. For this, nodes with identical labels, apart from the identifier, are merged, keeping the node with the smallest one, call it n, and deleting the others. All edges leaving from the latter nodes will be redirected to n. A graphoid built in this way is denoted a *minimal graphoid of* G, and it can be proved that it is unique. This minimisation process is denoted Minimise(G). For example, Figure 3 shows the minimal (Time.Year, Movie.All, Company.All, Actor.All)-graphoid for the base graph in Figure 1 (see also Example 3.3 below).

3.2 OLAP Operations on Graphs

We now review the OLAP operations over graphoids, which simulate the typical OLAP operations on cubes when they are represented as graphs.

Climbing and Aggregation. Let $\#\mathsf{n}_1, \ldots, \mathsf{n}_r$ be node types in a $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid G; and let $\#\mathsf{e}_1, \ldots, \mathsf{e}_s$ be edge types in G. The Climb$(G, \{\#\mathsf{n}_1, \ldots, \#\mathsf{n}_r, \#\mathsf{e}_1, \ldots, \#\mathsf{e}_s\}, D_k.(\ell_k \to \ell'_k))$ operation along the dimension D_k from level ℓ_k to level ℓ'_k in all nodes and edges of type $\#\mathsf{n}_i$ and $\#\mathsf{e}_i$, respectively, replaces any attribute value a from $dom(D_k.\ell_k)$ by the new value $\rho_{\ell_k \to \ell'_k}(a)$ from $dom(D_k.\ell'_k)$, in all nodes (edges) of G of types #n (#e), leaving G unaltered otherwise. Intuitively, the granularity of the graph is modified along D_k.

Example 3.3. A climbing operation to the Year level along dimension *Time*, and three climbs to the All level, along dimensions *Movie*, *Actor*, and *Company*, produce the (Time.Year, Movie.All, Company.All, Actor.All)-graphoid shown in Figure 4. Its minimal graphoid is the one in Figure 3. □

Consider a minimal $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid G, and a dimension D_k that appears in the hyperedges of G of type #e, and that plays the role of a measure, to which the aggregate function F_k can be applied. The *aggregation of* G *over* D_k *(using* F_k*)*, denoted Aggr$(G, \#\mathsf{e}, D_k, F_k)$, returns a graphoid G' over the same sets of nodes and edges, built as follows: If the hyperedges $e_1, e_2, \ldots, e_r$ are all of the same type and the nodes and edges agree in all attributes except (possibly) from an identifier attribute (and except from the dimension D_k), then $e_1, e_2, \ldots, e_r$ are replaced by one of them (say e_1) of the same type and with the same attribute values, apart from

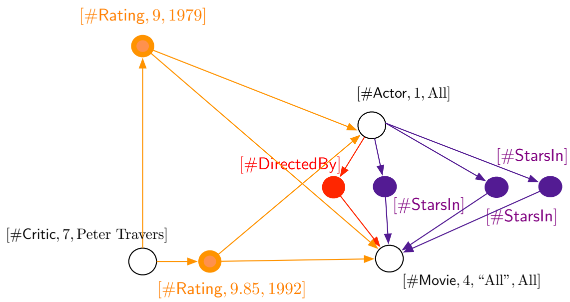

Figure 5: Roll-Up to Year, Actor, Movie, and Company for data of Figure 1.

the identifier. The value of $D_k.\ell_k$ becomes the value of the function F_k applied to the values of $D_k.\ell_k$ in the edges $e_1, e_2, ..., e_r$. To aggregate multiple dimensions $M_1, ..., M_k$, using functions $F_1, ..., F_k$, simultaneously, we write $\mathsf{Aggr}(G, \#\mathsf{e}, \{M_1, ..., M_k\}, \{F_1, ..., F_k\})$. We are now ready to define the well-known roll-up operation.

Roll-Up . Let G be a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid, a dimension D_c, and measure dimensions $M_1, ..., M_k$ that appear in the hyperedges of type #e of G, associated with the aggregate functions $F_1, ..., F_k$; Also, let $\#\mathsf{n}_i$ and $\#\mathsf{e}_i$ be node and hyperedge types appearing in G. *The roll-up of G over dimensions $M_1, ..., M_k$ (using functions $F_1, ..., F_k$), along the climbing dimension D_c from level ℓ_c to level ℓ'_c in nodes of types $\#\mathsf{n}_1, ..., \#\mathsf{n}_r$ and edges of types $\#\mathsf{e}_1, ..., \#\mathsf{e}_s$*, called $\mathsf{Roll\text{-}Up}(G, \{\#\mathsf{n}_1, ..., \#\mathsf{n}_r, \#\mathsf{e}_1, ..., \#\mathsf{e}_s\}, D_c.(\ell_c \to \ell'_c); \#\mathsf{e}, M_1, ..., M_k, F_1, ..., F_k)$, is defined as: $\mathsf{Aggr}(\mathsf{Minimise}(\mathsf{Climb}(G, \{\#\mathsf{n}_1, ..., \#\mathsf{n}_r, \#\mathsf{e}_1, ..., \#\mathsf{e}_s\}, D_c.(\ell_c \to \ell'_c))), \#\mathsf{e}, M_1, ..., M_k, F_1, ..., F_k)$.

Example 3.4. The operation $\mathsf{Roll\text{-}Up}(G, \{\#\mathsf{Rating}\}, \mathsf{Time}.(\mathsf{Day} \to \mathsf{Year}); \#\mathsf{Rating}, \mathsf{score}, \textsc{Avg})$ over the graphoid of Figure 3, produces the graphoid of Figure 5. The aggregated hyperedges are the two ones in Figure 3 that contain a #Rating node with Year=1992, and scores 9.7 and 10, respectively. □

The Drill-Down operation does the opposite of Roll-Up, taking a graphoid to a finer granularity level, along a dimension D_d. Descending from a level ℓ_d down to a level ℓ'_d along D_d is equivalent to climbing from the bottom level to the level ℓ'_d along D_d. Thus, we do not discuss this operation further here.

Dice. Given a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid, and a Boolean formula φ, a Boolean combination of atomic conditions of the form: (a) $D.\ell < c$, $D.\ell = c$ and $D.\ell > c$, where D is a dimension, ℓ is a level in that dimension, and $c \in dom(D.\ell)$; (b) $m < c$, $m = c$ and $m > c$, where m is a measure and c as in (a). $\mathsf{Dice}(G, \varphi)$, *produces a subgraphoid of G, whose nodes are the nodes of G and whose edges satisfy φ.* When an edge does not satisfy φ, the whole hyperedge is deleted. All other edges of G belong to $\mathsf{Dice}(G, \varphi)$.

Example 3.5. Applying $\mathsf{Dice}(G, \mathsf{Actor.Name} = \text{“Hoffman”})$ to the graphoid in Figure 1, produces the one in Figure 6 (left). □

Slice. The Slice operation on cubes, drops a dimension D_s, and aggregates all measures over D_s. We first need to roll-up to *All* along D_s, such that its domain is a singleton. On graphoids, the slice operation is thus defined as a roll-up to $D_s.All$ as follows. Given a graphoid G, a dimension D_s that appears in some nodes and/or hyperedges of G, measure dimensions $M_1, ..., M_k$ that appear in the hyperedges of G, and aggregate functions $F_1, ..., F_k$ associated with them; The *slice of the dimension D_s from G*, denoted $\mathsf{Slice}(G, D_s; M_1, ..., M_k, F_1, ..., F_k)$, is defined as $\mathsf{Roll\text{-}Up}(G, *, D_s.(\ell_s \to All); *, M_1, ..., M_k, F_1, ..., F_k)$.

Example 3.6. Applying $\mathsf{Slice}(G, \mathsf{Movie}; \mathsf{Score}, \textsc{Avg})$, and $\mathsf{Slice}(G, \mathsf{Company}; \mathsf{Score}, \textsc{Avg})$ to the graphoid of Figure 1, produces the graphoid of Figure 6 (right). □

The following theorem, proved in [12], supports our proposal.

Theorem 3.7. The cube OLAP-operations Roll-Up, Drill-Down, Slice and Dice can be expressed by OLAP-operations on graphoids.

4 IMPLEMENTATION

We are now ready to present our proof-of-concept implementation of the hypergraph MD model, and the OLAP operators of Section 3.2. For this, we chose the widely used Neo4J property graph database.[3] Although there are other options, like the orientdb graph database,[4] at the moment of writing this paper, theoretical research on Neo4J's declarative query language Cypher is being carried out, and there is a large number of users and developers working on Neo4j.

4.1 Dimensions and Graphoids

We next describe the representation of the background information, and how the base graph and the graphoids are implemented. The examples in this section are based on the case study that will be discussed in detail in Section 5.

4.1.1 Background information. Dimension schemas are represented as trees whose nodes are dimension levels. These nodes have two labels: the string *DimSchema*, and the name of the dimension schema; and a property called *level*, along with its value. For example, the schema of dimension *GeoPerson* (representing persons) is the tree containing nodes (*DimSchema*, *GeoPerson*, {level:Person}), (*DimSchema*, *GeoPerson*, {level:Country}), and (*DimSchema*, *GeoPerson*, {level:All}), shown on the left-hand side of Figure 7.

Dimension instances are also represented as trees, whose nodes are members of dimension levels. These nodes are connected according to the information in the dimension schema. When the Extraction, Transformation and Loading (ETL) process (that takes source data to the graph database) reads the information of level members, it creates the nodes, connects them to each other, and validates these links against the dimension schema. Moreover, since the new nodes store the information of the level to which they belong, this information can be used while processing the OLAP Operations without accessing the schema. A dimension instance node contains two labels: *DimInstance*, and the name of the dimension. There are also two property-value pairs: one for the for the level name (called *level*), and another one for the member value (called *value*). The right-hand side of Figure 7 illustrates this, showing a portion of the dimension instance for the dimension *GeoPerson*.

We reuse nodes whenever possible. That is, if a node is used multiple times, it is created only once, together with as many *from*/*to* links as needed. This strategy does not only save space, but also makes navigation more efficient. For example, in a dimension schema with multiple hierarchies, bottom and top levels are created

[3] http://neo4j.org
[4] http://orientdb.com/

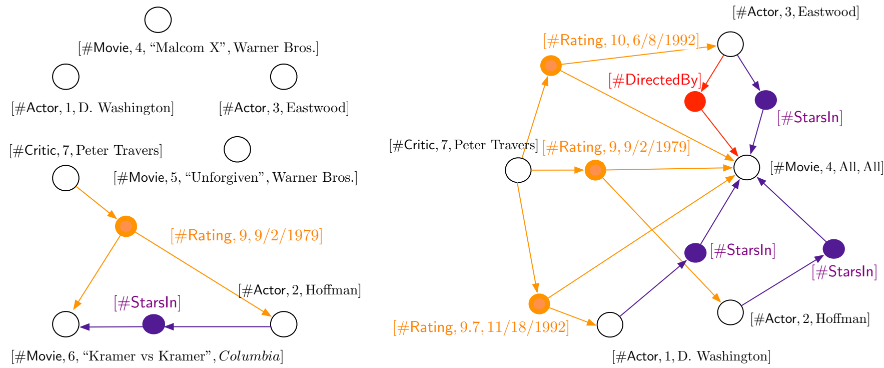

Figure 6: Dicing the graph for data about Dustin Hoffman for data on the right of Figure 1 (left); Slicing the Movie and Company dimensions for the data on the right of Figure 1.

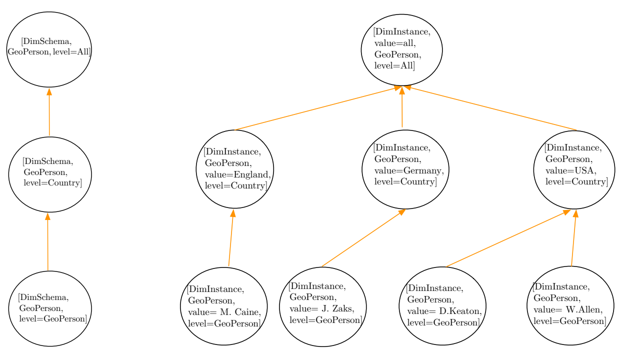

Figure 7: Schema representation for *GeoPerson* (left); An instance of *GeoPerson* (right).

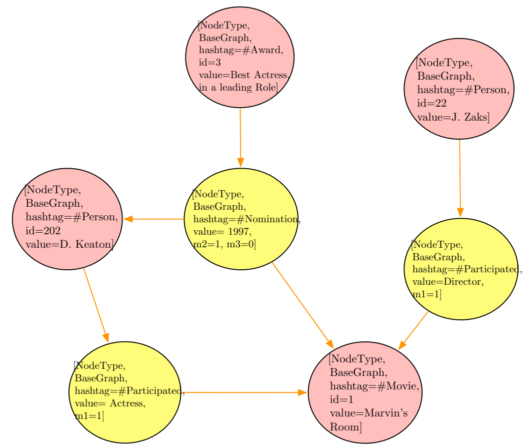

Figure 8: Representing a graphoid with Neo4j.

only once and used multiple times. The same idea is used for dimension instances. This is somehow similar to a snowflake schema in relational data warehouses (DW), where tables representing dimension levels are linked via foreign keys.

Given that dimension hierarchies are graphs, the choice of representing them using the same kind of data structure arises naturally. Since facts are graphs, this choice also prevents impedance mismatch. The latter would appear for example, when storing dimensions in a relational structure.

4.1.2 Implementation of base graphs and graphoids. Graphoids are multi-hypergraphs. However, Neo4J (like most graph databases) supports only binary relationships. Thus, we decided to represent both, node and edges types, as nodes, where “edge type” nodes are used to connect any number of node types. The direction of these relationships depends on the application. This way of modelling the hypergraph resembles the RDF graph data model, where blank nodes, typically used to represent n-ary relationships, are in some sense analogous to the “edge type” nodes. Another option could be to use a native hypergraph database (e.g., HypergraphDB[5]), which are not mature technologies so far. On the other hand, RDF triple stores have the drawback of not using native graph storage.

Figure 8 shows a small portion of the base graphoid of our case study (explained below). There are two edge types, #Participated and #Nomination. The former binds #Person and #Movie node types. The latter binds #Person, #Movie and #Award node types. A #Nomination edge type has an incoming edge from an #Award node type, and two outgoing edges to other node types. Again, we reuse nodes whenever possible. Node and edge types are labelled with the terms *NodeType* and *EdgeType*, respectively, and a unique name that indicates to which graph the nodes belong. This name can be the base graph (like in Figure 8), or a new graph produced by a query (see Section 4.2). Properties are also used to represent dimensions or measures (in the case of edge types).

4.2 Implementation of the OLAP Operations

We next describe the implementation of the operations presented in Section 3.2. Queries are expressed as a sequence of graph OLAP operations. The base graphoid is labelled *BaseGraph*. All the other graphoids are labelled with a new name created on-the-fly. Thus, the Neo4J database is composed of as many graphoids as queries were posed, each labelled with a different name. Graph OLAP operations first clone the graphoid over which they are posed; the new graphoid is then updated according to the operation's semantics. For instance, a Dice operation removes nodes and arcs, and a Rollup changes properties and merges nodes. As an example, the query:

[5]http://hypergraphdb.org/

```
Q1 <- Rollup(Nomination, Award->Organization, sum, sum)
```

first creates a graphoid labelled Q1, containing the information of the graphoid *Nomination*, except for the label. Q1 is then modified according to the semantics of the Rollup operation.

OLAP queries over graphs produce a sequence of *immutable* graphoids. In other words, each OLAP operation does not transform the original graphoid, but generates a new immutable one. Since we are aimed at addressing scenarios with multiple users querying the same original graphoid, an optimizer may take advantage of the intermediate graphoids and avoid recomputing them in different queries (graphoids which are rarely used could be removed from the system). In some sense this reminds Apache Spark's RDDs (Resilient Distributed Dataset),[6] that are immutable collections of elements, such that each transformation operation over an RDD generates a new one. Taking advantage of graphoid materialization for optimization is outside the scope of this paper.

Neo4j comes with a pattern-based query language named Cypher, and provides an API for embedding, e.g., a Java program. The API allows managing the underlying database and executing Cypher queries. The OLAP operators were implemented using this API interface. For example, the API: Rollup(DBConn, "Nomination","Q1", "Award","Award","Organization", SUM, SUM), where DBConn is an object that encapsulates the database connection settings, implements the query above. The strings "Nomination" and "Q1" are the labels of the input and output graphoids, respectively. The next three arguments correspond to the dimension, source level and target level of the roll-up operation. Finally, SUM is a Java interface that implements the sum operation.

5 CASE STUDY

We now present a case study that uses a portion of the Internet Movie Database (IMDB). *The problem consists in the analysis of prizes and nominations for people working in film making across t ime.* We aim at showing that, for some cases, representing the problem as graphs leads to a more natural and powerful representation than traditional OLAP modelling using cubes (in particular, we focus on Relational OLAP). For example, we will show that facts involving a variable number of dimensions and measures can be represented (and therefore analyzed) in a natural and flexible way. This is a well-known problem in classic OLAP, quite difficult and inefficient to represent in the typical star or snowflake models, usually leading to complex implementations [14]. *We remark that our intention at this time is to study how OLAP techniques can enhance graph analytics. Query optimization and performance are thus left for future work*, and we do not present here results for query execution times, since they would not be representative of actual performance. Also, for the sake of space, we do not include the details of the algorithms.

5.1 Classic OLAP Modelling

The following dimensions are defined as background information: *Movie*, with hierarchy $Movie \rightarrow All$; *GeoPerson*, with hierarchy $Person \rightarrow Country \rightarrow All$; *Role*, with hierarchy $Role \rightarrow All$; $Time$, with hierarchy $Year \rightarrow All$; and *Award*, with hierarchy $Award \rightarrow Organization \rightarrow All$. As an example of an (instance) path in the *Person* hierarchy, we have $Woody\ Allen \rightarrow USA \rightarrow all$; an example of a path instance in the *Award* hierarchy is $Best\ Actress\ in\ a\ Leading\ Role \rightarrow OscarAward \rightarrow all$.

[6] http://spark.apache.org

In traditional MD modelling based on facts and dimensions, a possible solution would be a model based on two fact tables: (a) one to represent the roles in which a person participated in a movie, e.g., with schema $Participation(Movie, Person, Role, Participates)$, where the measure $Participates$ represents the occurrence of a participation; and (b) one to represent nominations of a person for an award on a certain year, with schema, $Nomination(Movie, Person, Award, Year, Nominated, Won)$, where the measure Won tells if the award was obtained or not. We next show some example queries over this DW. Queries are expressed using a "data type agnostic" query language denoted Cube Algebra [4, 10, 11]. This language allows querying a data cube regardless its underlying data structure.

Query 1. *"Number of movies where Woody Allen participated as an actor".*

The query reads in Cube Algebra:

```
Q1 <- Dice(Participation, Person = 'Woody Allen')
Q2 <- Dice(Q1, Role='Actor')
Q3 <- Slice(Q2, Role, count)
Q4 <- Slice(Q3, Movie, count)
```

Here the result will be a one-dimensional cube, with one cell containing the actor's name and the number of movies. Note that *Person* was not sliced out. Thus, in a relational representation, the result will be a two-column table.

Query 2. *"Number of Oscar nominations and prizes by movie".*

This is expressed in Cube Algebra as:

```
Q5 <- Rollup(Nomination, Award->Organization, sum, sum)
Q6 <- Slice(Q5, GeoPerson, sum, sum)
Q7 <- Dice(Q6, Award.Organization = 'Oscar Award')
Q8 <- Slice(Q7, Time, sum, sum)
```

The result will contain, e.g., the tuple (or, in MD jargon, 'cell') $(Manhattan, OscarAward, 2, 0)$, since the movie was nominated for two Oscars, winning none of them. The resulting cube will contain two dimensions, since *GeoPerson* and $Time$ are sliced out.

Finally, we show a rather more complex query, involving the two cubes (or fact tables).

Query 3. *"Pairs of Movies and Persons, such that only people who played more than one role in it (other than Director), participated, listing only persons who were nominated for an Oscar in that movie, but did not win the award".*

This is expressed as:

```
Q9  <- Dice(Participacion, Role<>'Director')
Q10 <- Slice(Q9, Role, SUM)
Q11 <- Dice(Q10, Participates > 1 )
Q12 <- RollUp(Nomination, Award->Organization,SUM,SUM)
Q13 <- Dice(Q12, Award.Organization = 'Oscar Award'
       AND Won=0)
Q14 <- Slice(Q13, Award, SUM, SUM)
Q15 <- Slice(Q14, Year, SUM, SUM )
Q16 <- DrillAcross(Q11, Q15)
```

Here, two Cubes are queried: Participation, to compute the multiple roles played by a person in a movie, excluding the Director role; and Nomination, to find people and movies nominated to the Oscars, but who did not won it. Finally, a Drill Across operation between both cubes is performed. Behind the scenes, this operation is translated into an expensive join operator between two fact tables.

5.2 OLAP Modelling of Graphs

We now address the problem above using graph OLAP instead of the classic solution. We consider the same dimensions and hierarchies as in Section 5.1. There are three node types, each one corresponding to a background dimension: #Movie, #Person, and #Award. Each node type is associated with a dimension as follows: (a) (#Movie, *id*, *Movie*); (b) (#Person, *id*, *GeoPerson*); (c) (#Award, *id*, *Award*). Two edge types are also defined: (a) (#Participated, *Role*, $m1$); (b) #Nomination, *Time*, $m2$, $m3$). Type #Participated indicates who participated in a movie, and in which role. Measure $m1$ is analogous to *Participates* in the OLAP cube *Participation* of Section 5.1. The edge type #Nomination links a person with a movie and an award. Measures $m2$ and $m3$ represent the number of nominations and of successful nominations, respectively, similarly to both measures in the OLAP cube *Nomination*. Note the use of dimensions *Role* and *Time* in the graph representation. These dimensions here appear in the edge types, rather than as node types. Figure 9 shows a small portion of the IMDB database, represented as a Neo4j graph. Let us consider the #Person node for Woody Allen (on the left of the figure). We can see that he performed and directed the movie "Annie Hall" (there are two #Participated hyperedges, one with the role attribute *Director*, and the other with the role attribute *Actor*). We can also see (close to the top-left part of the figure) two #Nomination hyperedges, composed of the node types #Award, #Person, and #Movie. These indicate the Oscar and BAFTA awards for best direction. The Oscar nomination is represented by the hyperedge with identifier '116', while the node identifier for the BAFTA nomination is '124'. The graph also shows that Diane Keaton was nominated (and won) the "Best Actress in a Leading Role" Oscar, and the "Best Actress" BAFTA awards. Note that the two cubes in the previous section are now represented in a single graph, which is more natural, and closer to the real world situation. We show next how the queries above are expressed over this graph and the background information.

5.3 OLAP queries over Graphoids

Query 1. *"Number of movies in which Woody Allen participated as an actor".*

The base graphoid is named *BaseGraph*. Thus, the query, is written as the following sequence of function calls:

```
Operators.Dice(graphDB,"BaseGraph","Q1","GeoPerson",
               "GeoPerson", "=", "Woody Allen" );
Operators.Dice(graphDB,"Q1","Q2","Role","Role","=",
               "Actor");
Operators.Slice(graphDB, "Q2","Q3","Role",
               Arrays.asList("m1","m2","m3"),
               Arrays.asList(COUNT,COUNT,COUNT));
Operators.Slice(graphDB,"Q3","Q4","Movie",
               Arrays.asList("m1","m2","m3"),
               Arrays.asList(COUNT,COUNT,COUNT));
```

Note that the sequence of API calls replicates the sequence of Cube Algebra operations for the queries in Section 5.1, so the API details could be easily hidden. Figure 10 (left) shows a portion of the resulting graphoid, telling the number of movies in the sample database where Woody Allen performed.

Query 2. *"Number of Oscar nominations and prizes by movie".*

```
Operators.Rollup(graphDB,"BaseGraph","Q5",
       Arrays.asList({#Award}),
       Award.(Award->Organization),
       "#Nomination", Arrays.asList("m1", "m2", "m3"),
       Arrays.asList(SUM,SUM,SUM));
Operators.Slice(graphDB,"Q5","Q6","GeoPerson",
       Arrays.asList("m1","m2","m3"),
       Arrays.asList(SUM,SUM,SUM));
Operators.Dice(graphDB,"Q6","Q7","Award",
       "Organization", "=", "Oscar Award");
Operators.Slice(graphDB,"Q7","Q8","Year",
       Arrays.asList("m1","m2","m3"),
       Arrays.asList(SUM,SUM,SUM));
```

A portion of the result is shown in Figure 10 (center). We can see information of four movies. "Cafe Society" appears as an isolated node, because it did not receive any Oscar nomination. The node type #Person was sliced out by the query, thus, it appears with value "all". The node type #Award appears with the value "*Oscar*", since this node was rolled-up to the level *Organization* and later diced. The edge type #Nomination appears with attribute "all" on the *Time* dimension. Its measures show the number of (summarized) nominations and prizes. For instance, "Manhattan" received two Oscar nominations but won none of them, at it is shown in the leftmost path in the figure.

Query 3. *"Pairs of Movies and Persons, such that only people who played more than one role in it (other than Director) are considered, listing only persons who were nominated for an Oscar in that movie, but did not win the award".*

```
Operators.Dice(graphDB,"BaseGraph","Q9","Role","Role",
        "<>", "Director");
Operators.Slice(graphDB,"Q9","Q10","Role",
        Arrays.asList("m1","m2","m3"),
        Arrays.asList(SUM,SUM,SUM));
Operators.Dice(graphDB,"Q10","Q11","measures","m1",
        "<>","1.0");
Operators.Rollup(graphDB,"Q11","Q12",
        Arrays.asList("#Award"),
       "Award", "Award", "Organization","#Nomination",
        Arrays.asList("m1","m2","m3"),
        Arrays.asList(SUM,SUM,SUM));
Operators.Dice(graphDB,"Q12","Q13","Award",
        "Organization","=", "Oscar Award" );
Operators.Dice(graphDB,"Q13","Q14","measures",
        "m3","=","0");
Operators.Slice(graphDB, "Q14","Q15","Award",
        Arrays.asList("m1","m2","m3"),
        Arrays.asList(SUM,SUM,SUM));
Operators.Slice(graphDB, "Q15","Q16","Year",
        Arrays.asList("m1","m2","m3"),
        Arrays.asList(SUM,SUM,SUM));
```

Here we can clearly see the advantage of the graph approach over the relational OLAP one. The query is answered navigating a single graph, avoiding joining tables (that means, the join is performed through navigation, in a more efficient way), or drilling across two or more cubes (which is required by the representation shown in Section 5.1). In a real-world 'Big Data" setting, this can also simplify the ETL process, especially when source data come as unstructured data. Figure 10 (right) shows part of the result.

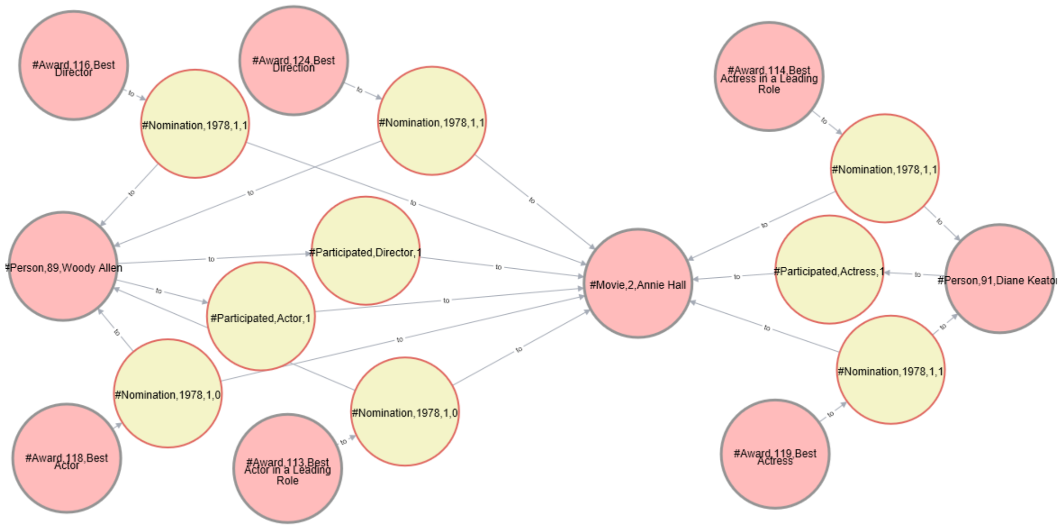

Figure 9: IMDB use case in Graph OLAP.

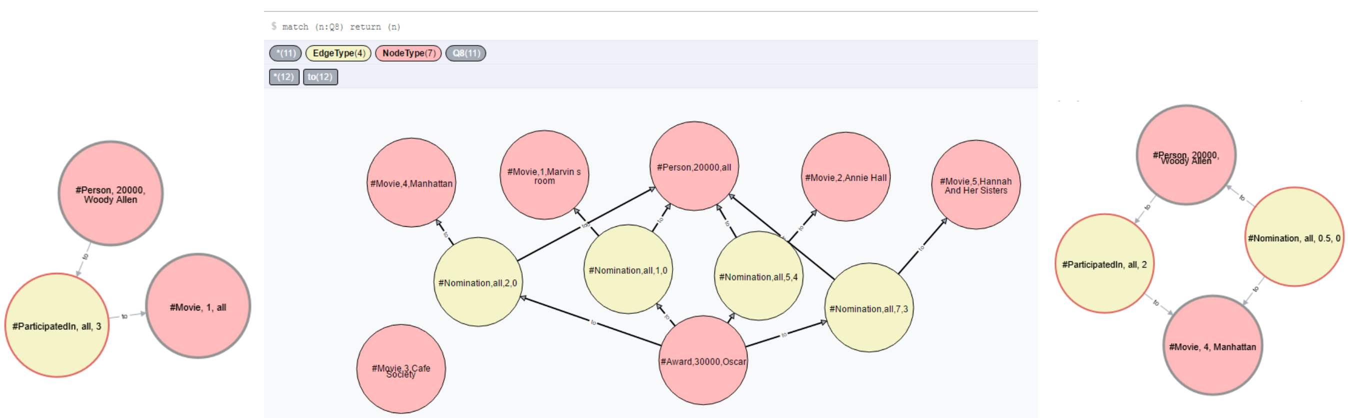

Figure 10: Portion of the results for Q1 (left), Q2 (center), and Q3 (right), in the Neo4J interface.

6 CONCLUSION AND FUTURE WORK

We have presented a proof-of-concept implementation, over a Neo4j database, of a MD data model for graph analysis and its associated OLAP operators. We also discussed a case study, showing that modelling an analysis problem as graphs may lead to a more natural and efficient solution. Our next steps will focus on efficiency, and, for that, we think on using the graphoids as materialized views, along the lines suggested in [15].

Acknowledgments. Alejandro Vaisman was supported by a travel grant from Hasselt University (Korte verblijven–inkomende mobiliteit, BOF16KV09), and by the PICT-0787-2014 project.

REFERENCES

[1] R. Angles. A Comparison of Current Graph Database Models. In *Proceedings of ICDE Workshops*, pages 171–177, 2012.
[2] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1):1:1–1:39, 2008.
[3] C. Chen, X. Yan, F. Zhu, J. Han, and P. Yu. Graph OLAP: a multi-dimensional framework for graph data analysis. *Knowl. Inf. Syst.*, 21(1):41–63, 2009.
[4] C. Ciferri, R. Ciferri, L. Gómez, M. Schneider, A. Vaisman, and E. Zimányi. Cube algebra: A generic user-centric model and query language for OLAP cubes. *IJDWM*, 9(2):39–65, 2013.
[5] B. Denis, A. Ghrab, and S. Skhiri. A distributed approach for graph-oriented multidimensional analysis. In *IEEE Big Data*, pages 9–16, 2013.
[6] A. Ghrab, O. Romero, S. Skhiri, A. Vaisman, and E. Zimányi. GRAD: Modeling and Querying Data Warehouses. In *Proceedings of ADBIS*, pages 92–105, 2015.
[7] O. Hartig. Reconciliation of RDF* and property graphs. *CoRR*, abs/1409.3288, 2014.
[8] D. T. A. Hoang, T. Priebe, and A. M. Tjoa. Hypergraph-based multidimensional data modeling towards on-demand business analysis. In *Proceedings of iiWAS*, pages 36–43, 2011.
[9] R. Kimball. *The Data Warehouse Toolkit.* J. Wiley and Sons, 1996.
[10] B. Kuijpers and A. Vaisman. A formal algebra for OLAP. *CoRR*, abs/1609.05020, 2016.
[11] B. Kuijpers and A. Vaisman. An algebra for OLAP. *Intelligent Data Analysis*, 21(5), 2017.
[12] B. Kuijpers and A. Vaisman. OLAPing graph data. *Submitted manuscript*, 2017.
[13] Q. Qu, F. Zhu, X. Yan, J. Han, P. S. Yu, and H. Li. Efficient topological OLAP on information networks. In *Proceedings of DASFAA*, pages 389–403. Springer, 2011.
[14] A. Vaisman and E. Zimányi. *Data Warehouse Systems: Design and Implementation.* Springer, 2014.
[15] Z. Wang, Q. Fan, H. Wang, K.-L. Tan, D. Agrawal, and A. E. Abbadi. Pagrol: Parallel graph OLAP over large-scale attributed graphs. In *Proceedings of ICDE*, pages 496–507, 2014.
[16] P. Zhao, X. Li, D. Xin, and J. Han. Graph cube: on warehousing and OLAP multidimensional networks. In *Proceedings of SIGMOD*, pages 853–864. ACM, 2011.