Online analytical processsing on graph data

Leticia Gómez[a], Bart Kuijpers[b] and Alejandro Vaisman[a,*]
[a] *Instituto Tecnológico de Buenos Aires, Buenos Aires, Argentina*
[b] *Databases and Theoretical Computer Science Research Group, Data Science Institute, UHasselt – Hasselt University, Belgium*

Abstract. Online Analytical Processing (OLAP) comprises tools and algorithms that allow querying multidimensional databases. It is based on the multidimensional model, where data can be seen as a cube such that each cell contains one or more measures that can be aggregated along dimensions. In a "Big Data" scenario, traditional data warehousing and OLAP operations are clearly not sufficient to address current data analysis requirements, for example, social network analysis. Furthermore, OLAP operations and models can expand the possibilities of graph analysis beyond the traditional graph-based computation. Nevertheless, there is not much work on the problem of taking OLAP analysis to the graph data model.

This paper proposes a formal multidimensional model for graph analysis, that considers the basic graph data, and also background information in the form of dimension hierarchies. The graphs in this model are node- and edge-labelled directed multi-hypergraphs, called *graphoids*, which can be defined at several different levels of granularity using the dimensions associated with them. Operations analogous to the ones used in typical OLAP over cubes are defined over graphoids. The paper presents a formal definition of the graphoid model for OLAP, proves that the typical OLAP operations on cubes can be expressed over the graphoid model, and shows that the classic data cube model is a particular case of the graphoid data model. Finally, a case study supports the claim that, for many kinds of OLAP-like analysis on graphs, the graphoid model works better than the typical relational OLAP alternative, and for the classic OLAP queries, it remains competitive.

Keywords: OLAP, data warehousing, graph database, big data, graph aggregation

1. Introduction

Online Analytical Processing (OLAP) [15,26] comprises 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 sets of hierarchies. Traditional OLAP operations are used to manipulate the data cube, for example: aggregation and disaggregation of measure data along the dimensions; selection of a portion of the cube; or projection of the data cube over a subset of its dimensions. The cube is computed after a process called ETL, an acronym for Extract, Transform, and Load, which requires a complex and expensive load of work to carry data from the sources to the MD database, typically a data warehouse (DW). Although OLAP has been used for social network analysis [16,19], in a "Big Data" scenario, further requirements appear [7]. In the classic paper by Cohen et al. [6], the so-called MAD skills (standing from Magnetic, Agile and Deep) required for data analytics are described. In this scenario, more complex analysis tools are required, that go beyond classic OLAP [22]. Graphs, and,

[*] Corresponding author: Alejandro Vaisman, Instituto Tecnológico de Buenos Aires, Buenos Aires, Argentina. E-mail: avaisman@itba.edu.ar.

particularly, property graphs [13,20], are becoming increasingly popular to model different kinds of networks (for instance, social networks, sensor networks, and the kind). Property graphs underlie the most popular graph databases [1]. Examples of graph databases and graph processing frameworks following this model are Neo4j,[1] Janusgraph[2] (previously called Titan), and GraphFrames.[3] In addition to traditional graph analytics, it is also interesting for the data scientist to have the possibility of performing OLAP on graphs.

From the discussion above, it follows that, on the one hand, traditional data warehousing and OLAP operations on cubes are clearly not sufficient to address the current data analysis requirements; on the other hand, OLAP operations and models can expand the possibilities of graph analysis beyond the traditional graph-based computation, like shortest-path, centrality analysis and so on. In spite of the above, not many proposals have been presented in this sense so far. In addition, most of the existing work addresses homogeneous graphs (that is, graphs where all nodes are of the same type), where the measure of interest is related to the OLAP analysis on the graph topology [5,28,31]. Further, existing works only address graphs with binary relationships (see Section 2 for an in-depth discussion on these issues). However, real-world graphs are complex and often heterogeneous, where nodes and edges can be of different types, and relating different numbers of entities.

This paper proposes a 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 this model are node- and edge-labelled directed multi-hypergraphs, called *graphoids*. In essence, these can be denoted “property hypergraphs”. A graphoid can be defined at several different levels of granularity, using the dimensions associated with them. For this, the Climb operation is available. Over this model, operations like the ones used in typical OLAP on cubes are defined, namely Roll-Up, Drill-Down, Slice, and Dice, as well as other operations for graphoid manipulation, e.g., n-delete (which deletes nodes). The hypergraph model allows a natural representation of facts with different dimensions, since hyperedges can connect a variable number of nodes of different types. A typical example is the analysis of phone calls, the running example that will be used throughout this paper. Here, not only point-to-point calls between two partners can be represented, but also “group calls” between any number of participants. In classic OLAP [15], a group call must be represented by means of a fact table containing a fixed number of columns (e.g., caller, callee, and the corresponding measures). Therefore, when the OLAP analysis for telecommunication information concerns point-to-point calls between two partners, the relational representation (denoted ROLAP) works fine, but when this is not the case, modelling and querying issues appear, which calls for a more natural representation, closer to the original data format. And here is where the hypergraph model comes to the rescue [11]. In summary, the main contributions of the paper are:

1. A graph data model based on the notion of graphoids;
2. The definition of a collection of OLAP operations over these graphoids;
3. A proof that the classical OLAP operations on cubes can be simulated by the OLAP operations defined in the graphoid model and, therefore, that these graphoid-based operations are at least as powerful as the classical OLAP operations on cubes;
4. A case study and a series of experiments, that give the intuition of a class of problems where the graphoid model works clearly better than relational OLAP, whereas for classic OLAP queries, the graph representation is still competitive with the relational alternative.

[1] http://www.neo4j.com.

[2] http://janusgraph.org/.

[3] https://graphframes.github.io/.

In addition to the above, of course all the classic analysis tools from graph theory are supported by the model, although this topic is beyond the scope of this paper.

Remark 1. This paper does not claim that the graphoid model is always more appropriate than the classic relational OLAP representation. Instead, the proposal aims at showing that when a more flexible model is needed, where *n-ary* relationships between instances are present (and *n* is variable), the model allows not only for a more natural representation, but also can deliver better performance for some critical queries. □

The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 presents the graphoid data model. Section 4 presents the OLAP operations on graphoids, while Section 5 shows that the graphoid OLAP operations capture the classic OLAP operations on cubes. Section 6 discusses a case study and presents an experimental analysis. Section 7 concludes the paper.

2. Related work

The model described in the next sections is based on the notion of property graphs [2]. In this model, nodes and edges (hyperdeges, as will be explained later) are labelled with a sequence of attribute-value pairs. It will be assumed that the values of the attributes represent members of dimension levels (i.e., each attribute value is an element in the domain of a dimension level), and thus nodes and edges can be aggregated, provided that an attribute hierarchy is defined over those dimensions. Property graphs are the usual choice in modern graph database models used in practical implementations. Attributes are included in nodes and edges mainly aimed at improving the speed of retrieval of the data directly related to a given node. Here, these attributes are also used to perform OLAP operations.

2.1. Graph database modelling

There is an extensive bibliography on graph database models, comprehensively studied in [3]. Therefore, the interested reader is referred to this work. Multiple native graph indexing and query languages (e.g. GraphQL [14]) were developed to efficiently answer graph-oriented queries. More recently, Angles [1] compares modern graph database models underlying the most used graph databases. Such study is based on the data models (that is, data structure, query language, and integrity constraints), leaving out physical implementation issues. The study also shows that summarization is not considered a native property of these databases.

Two graph database models are used in practice:

(a) Models based on RDF,[4] oriented to the Semantic Web. This is the case of AllegroGraph,[5] the only graph database which supports reasoning at the time of writing this work.

(b) Models based on property graphs, introduced above.

Models of type (a) represent data as sets of triples where each triple consists of three elements that are referred to as the subject, the predicate, and the object of the triple. These triples allow describing arbitrary objects in terms of their attributes and their relationships to other objects. Informally, a collection of RDF triples is an RDF graph. Although the models in (a) have a general scope, the structure

[4] https://www.w3.org/RDF/.

[5] http://franz.com/agraph/allegrograph/.

of RDF makes them not as efficient as the other models, which are aimed at reaching a local scope. An important feature of RDF-based graph models, however, is that they follow a standard, which is not yet the case for the other graph databases. Hartig [13] proposes a formal way of reconciling both models formally, through a collection of well-defined transformations between property graphs and RDF graphs. He shows that property graphs could, in the end, be queried using SPARQL,[6] the standard query language for the Semantic Web.

In [4], GOLAP, a framework for OLAP on RDF data is introduced. The work presents a graph model for OLAP, and an extension to SPARQL (denoted FSPARQL) for writing OLAP queries over RDF data. GOLAP supports implicit relationships between nodes and different partitioning types of graph elements. Other solutions for OLAP modelling and querying on RDF and the Semantic Web can be found in [9,25], where the QB4OLAP vocabulary is the basis for defining the OLAP operations on RDF graphs.

It is remarked that the present paper is based on the property graph data model.

2.2. *Graph summarization and OLAP*

Graph summarization is a critical operation for multi-level analysis of graph data. Tian et al. [23] proposed the SNAP operation (standing for Summarization by Grouping Nodes on Attributes and Pairwise Relationships) to produce a summary graph by grouping nodes based on node attributes and relationships selected by the user. The authors introduce the k-SNAP operation, which allows drilling-down and rolling-up at different aggregation levels. The work also shows that the k-SNAP computation is NP-complete, and proposes two heuristic methods to approximate the k-SNAP results. The main difference with OLAP-style aggregation is that SNAP does not take into account rollup functions, but aggregates nodes based on the strength of the relationships between them. Therefore, although along the lines of graph summarization, this work does not strictly fit into what is typically denoted graph OLAP.

GraphOLAP [5] is a conceptual framework for performing OLAP over a collection of homogeneous graphs. Attributes of the snapshots are considered as the dimensions, and aggregations of the graph are performed by overlaying a collection of graph snapshots. Further, dimensions are classified as topological and informational. Informational OLAP aggregations consist in edge-centric snapshot overlaying. Thus, only edges change whereas no changes to the nodes are made. Topological OLAP aggregations consist in merging nodes and edges by navigating through the node hierarchy. Along the same lines, Qu et al. [18] introduced a more detailed framework for topological OLAP analysis of graphs. The authors discuss the structural aggregation of the graph, following the OLAP paradigm, presenting techniques based on the properties of the graph measures for optimizing measure computation at the different aggregation levels. GraphCube [31] is a framework for OLAP cubes computation and analysis at the different levels of aggregation of a graph. It targets single, homogeneous, node-attributed graphs. Also, the framework introduces so-called cuboid and crossboid queries, for building and analyzing the different graph cubes.

Pagrol [28] is a MapReduce framework for distributed OLAP analysis of homogeneous attributed graphs. Pagrol extends the model of GraphCube by considering the attributes of the edges as dimensions. The idea here is to efficiently compute all possible aggregations of an homogeneous graph, by means of parallel computation using the MapReduce programming paradigm. Although efficiency is achieved, from an analytical point of view, this proposal has some limitations, since it only addresses

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

roll-up and drill-down operations on homogeneous graphs. Also based on the notion of so-called graph cuboids, Distributed Graph Cube [8] proposes a distributed framework for graph cube computation and aggregation, implemented using Spark[7] and Hadoop.[8] Again, this proposal only supports *homogeneous* graphs. Wang et al. [27] aims at extending [28] to heterogeneous graphs, proposing TSMH Graph Cube, a framework for answering OLAP queries over heterogeneous graphs. Three query categories are addressed in this work: (a) Queries about information between entities; (b) Knowledge discovery on an aggregate graph; (c) Building new graphs using discovered relationships. Both works address the problem basically by building cuboids merging the graph using the node attributes, which represent the dimensions. In addition to roll-up and drill-down, the Attribute-Transformation operation is proposed, to create new types of nodes, also creating new edges between these new nodes. In another approach, and starting from the topological and informational dimensions introduced in [5], in order to extend this work to heterogenoeus graphs, Yin et al. [30] propose HMGraph OLAP, a data warehousing model based on the notion of entity dimensions. In this work, two basic operations are defined, namely, rotate and stretch, which allow defining different views of the same graph, defining entities as relationships and vice versa. Claiming to address the limitations of the works above, Wu et al. [29] introduce P&D Graph Cube, a model aimed at querying heterogeneous networks (typically social networks), not oriented to classic rolling-up and drilling-down on cubes, but to aggregating graphs based on the relationships between nodes. The idea here is to obtain insights from three kinds of queries: entity, edge, and topological structure query. The authors specifically express that the work does not address typical OLAP queries based on measure aggregation along dimensions.

Regarding OLAP graph modeling, in [10] the authors propose a framework for building OLAP cubes from graph data. The framework is aimed at extracting the candidate multidimensional spaces in heterogeneous property graphs, and, in general, at providing insight into the multidimensional concepts in graph data. In this model, only binary relationships between nodes are supported. Rudolf et al. [21] also propose a very preliminary multidimensional graph model for OLAP, aimed at capturing the classic notions of facts and dimensions.

A key difference between the works described above and the proposal introduced in this paper, is that the latter supports the notion of *OLAP hypergraphs*, highly expanding the possibilities of analysis. This way, instead of binary relationships between nodes, there are n-ary, probably duplicated relationships, which are typical in Data Warehousing and OLAP. Further, supporting n-ary relationships allows naturally modelling OLAP situations where different facts have a different number of relations, like in the group calls case commented in Section 1, and studied in Section 6. In other words, the model handles multi-hypergraphs. Also, the paper works over the classic OLAP operations, and formally defines their meaning in a graph context. This approach allows an OLAP user to work with the notion of a data cube at the conceptual level [26], regardless the kind of underlying data (in this case, graphs), defining OLAP operations in terms of cubes and dimensions rather than in terms of nodes and edges. Finally, the authors have shown the usefulness of this proposal in different scenarios, like trajectory analysis [12] and typical OLAP analysis on social networks [24].

3. Data model

This section presents the graphoid OLAP data model. First, background dimensions are formally defined, along the lines of the classic OLAP literature. Then, the (hyper) graph data model is introduced.

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

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

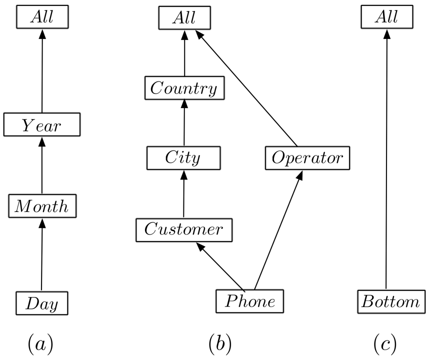


Fig. 1. Dimension schemas for the dimensions Time (a), Phone (b), and Id (identifier) (c).

3.1. Hierarchies and dimensions

The notions of dimension schema and dimension graph (or dimension instance) that will be used throughout the paper, are introduced first. These concepts are needed to make the paper self-contained, and to understand the examples. The reader is referred to [17] for full details of the underlying OLAP data model.

Definition 1 (Dimension schema, hierarchy and level)**.** Let D be a name for a dimension. A *dimension schema* $\sigma(D)$ *for* D is a lattice (a partial order), with a unique top-node, called *All* (which has only incoming edges) and a unique bottom-node, called *Bottom* (which has only outgoing edges), such that all maximal-length paths in the graph go from *Bottom* to *All*. Any path from *Bottom* to *All* in a dimension schema $\sigma(D)$ is called a *hierarchy* of $\sigma(D)$. Each node in a hierarchy (that is, in a dimension schema) is called a *level* of $\sigma(D)$. □

The running example used throughout this paper analyses calls between customers, which belong to different companies. For this, as background (contextual) information for the graph data representing calls (to be explained later), there is a Phone dimension, with levels Phone (representing the phone number), Customer, City, Country, and Operator. There is also a Time dimension, with levels Date, Month, and Year. The following examples explain this in detail.

Example 1. Figure 1 depicts the dimension schemas $\sigma(\textit{Phone})$ and $\sigma(\textit{Time})$, for the dimensions Phone and Time, respectively. In addition, there is also a dimension denoted Id, representing identifiers, that will be explained later. In the dimension Phone, it holds that *Bottom* = Phone, and there are two hierarchies denoted, respectively, as

Phone → Customer → City → Country → *All*,

and

Phone → Operator → *All*.

The node Customer is an example of a level in the first of the above hierarchies. For the dimension Time, *Bottom* = Day holds, as well as the hierarchy Day → Month → Year → *All*. □

Definition 2 (Level, hierarchy, and dimension instances)**.** Let D be a dimension with schema $\sigma(D)$, and let ℓ be a level in $\sigma(D)$. A *level instance of* ℓ is a non-empty, finite set $\textit{dom}(D.\ell)$. If $\ell = \textit{All}$, then $\textit{dom}(D.\textit{All})$ is the singleton $\{\textit{all}\}$. If $\ell = \textit{Bottom}$, then $\textit{dom}(D.\textit{Bottom})$ is the domain of the dimension D, that is, $\textit{dom}(D)$.

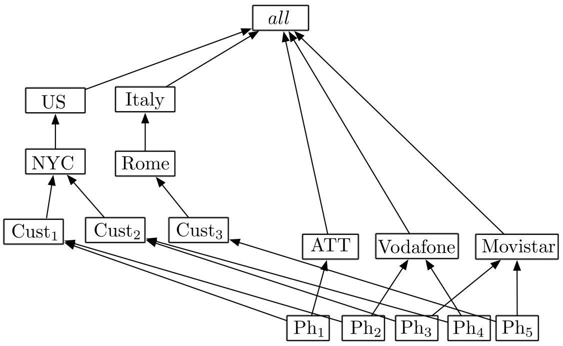


Fig. 2. An example of a dimension instance $I(\sigma(\mathit{Phone}))$ for the dimension *Phone*.

A *dimension graph (or instance)* $I(\sigma(D))$ over the dimension schema $\sigma(D)$ is a directed acyclic graph with node set

$$\bigcup_{\ell} \mathit{dom}(D.\ell),$$

where the union is taken over all levels in $\sigma(D)$. The edge set of this directed acyclic graph is defined as follows. Let ℓ and ℓ' be two levels of $\sigma(D)$, and let $a \in \mathit{dom}(D.\ell)$ and $a' \in \mathit{dom}(D.\ell')$. 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* (relative to the dimension instance $I(\sigma(D))$) is the subgraph of $I(\sigma(D))$ with nodes from $\mathit{dom}(D.\ell)$, for ℓ appearing in H. This subgraph is denoted $I_H(\sigma(D))$. □

Remark 2. A hierarchy instance $I_H(\sigma(D))$ is always a (directed) tree, since a hierarchy is a linear lattice. The following terminology is used. 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 it is said that a *rolls-up* to b, and denoted by $\rho_H(a, b)$ (or $\rho(a, b)$ if H is clear from the context). Example 2 illustrates these concepts. □

Example 2. Consider dimension Phone whose schema $\sigma(\mathit{Phone})$ is given in Fig. 1b. Associated with this schema, there is an instance where $\mathit{dom}(\mathit{Phone}) = \mathit{dom}(\mathit{Phone.Bottom}) = \mathit{dom}(\mathit{Phone.Phone}) = \{Ph_1, Ph_2, Ph_3, Ph_4, Ph_5\}$. Also, at the Operator level, $\mathit{dom}(\mathit{Phone.Operator}) = \{\mathit{ATT}, \mathit{Movistar}, \mathit{Vodafone}\}$. This dimension instance $I(\sigma(\mathit{Phone}))$ is depicted in Fig. 2, which shows, e.g., that phone lines Ph_2 and Ph_4 correspond to the operator *Vodafone*. □

In what follows, "sound" dimension graphs are assumed. In thses graphs, rolling-up from the *Bottom* level, to the same element along different paths, gives the same result [17], typical in so-called balanced (or homogeneous) dimensions [26].

3.2. The base graph and graphoids

As a basic data structure for modelling OLAP on graph data, the concept of *graphoid* is introduced and defined in this section. A graphoid plays the role of a multi-dimensional cuboid in classical OLAP and it is designed to represent the information of the application domain, at a certain level of granularity. Essentially, *a graphoid is a node- and edge-labelled directed multi-hypergraph.*

In what follows, a collection of dimensions $D_1, \ldots, D_d$ is assumed in the application domain, and their schemas $\sigma(D_1), \ldots, \sigma(D_d)$ are given. Furthermore, hierarchy instances $I(\sigma(D_1)), \ldots, I(\sigma(D_d))$

for all dimensions are given. Finally, assume that a special dimension $D_0 = \text{Id}$ is given, to represent unique identifiers (Fig. 1c). The notions of *attributes*, *node types* and *edge types* are defined next.

3.2.1. Attributes

The set of attributes $\mathcal{A}$ that describe the data is defined as $\mathcal{A} = \{D.\ell \mid D \in \{D_0, D_1, \ldots, D_d\}$ and ℓ is a level of $D\}$. As described in Section 3.1, to each attribute A of $\mathcal{A}$, a *domain* $\mathit{dom}(A)$ is associated, from which the attribute takes values.

3.2.2. Node types

Assume a finite, non-empty set $\mathcal{N}$ of *node types*. Elements of $\mathcal{N}$ are denoted by a string starting with a hashtag. For example, the node type #Phone indicates that a node in a graph represents a phone line number. There are also two functions, ar and *dim* defined on $\mathcal{N}$. For each node type #n in $\mathcal{N}$, $ar(\#\text{n})$ is a natural number, called the *arity*, that expresses the number of attributes associated with a node of type #n. Also, *dim*(#n) is an $ar(\#\text{n})$-tuple of attributes, which are dimensions defined at the *Bottom* level, the first of which is the Identifier dimension. This means that *dim*(#n) is an element of $\{\text{Id}\} \times \{D_1, \ldots, D_d\}^{ar(\#\text{n})-1}$. The tuple *dim*(#n) tells which attributes are associated with a node of type #n, without specifying their levels. Finally, assume that *dim*(#n) contains no repetition, which is the usual case in practice. The identifier dimension is always used at its *Bottom* level.

3.2.3. Edge types

Assume the existence of a finite, non-empty set $\mathcal{E}$ of *edge types*, which is disjoint from the set $\mathcal{N}$. Elements of $\mathcal{E}$ will also be denoted by a string starting with a hashtag. For example, the node type #Call indicates that an edge connects nodes that participate in a call. Again, also assume the existence of the functions ar and *dim* on $\mathcal{E}$. To each edge type #e in $\mathcal{N}$, $ar(\#\text{e})$ is a natural number, called the *arity*, that expresses the number of attributes associated with an edge of type #e. Also, *dim*(#e) is an $ar(\#\text{e})$-tuple of attributes, which are dimensions (at Bottom-level). This means that *dim*(#e) is an element of $\{D_0, D_1, \ldots, D_d\}^{ar(\#\text{e})}$. The tuple *dim*(#n) expresses which attributes are associated with an edge of type #e, without specifying their levels. Finally, assume that *dim*(#e) contains no repetition. The identifier dimension (at its *Bottom* level) may appear, but is not required. If the identifier dimension appears, this only occurs once, among the attributes that describe edges of a certain type.

It is now possible to define the notion of *graphoid.*

Definition 3 (Graphoid)**.** Let $D_0 = \text{Id}$ be the identifier dimension. Let dimensions $D_1, \ldots, D_d$ be given with their respective schemas and instances. Let $\ell_1, \ldots, \ell_d$ be levels for these respective dimensions. A $(D_1.\ell_1, \ldots, D_d.\ell_d)$*-graphoid* (or *graphoid*, for short, if the levels are clear from the context) is a 6-tuple $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$, where

- N is a finite, non-empty set, called the set of *nodes* of G;
- τ_N is a function from N to $\mathcal{N}$ (that associates a unique type with each node of G);
- λ_N is a function that maps a node $n \in N$ to a string $[\#\text{n}, a_1, \ldots, a_{ar(\#\text{n})}]$, where $\#\text{n} = \tau_N(n)$ and, if *dim*(#n) $= (A_1, \ldots, A_{ar(\#\text{n})})$, then, for $i = 1, \ldots, ar(\#\text{n})$, $a_i \in \mathit{dom}(D_j.\ell_j)$, if A_i is the dimension D_j. It is assumed that different a_1-values are associated with different nodes, since the first attribute value acts as a node identifier; λ_N is denoted the *node labelling function*;
- E is a subbag[9] of the set $\mathcal{P}(N) \times \mathcal{P}(N)$, which we call the set of *(multi hyper-)edges* of G;

[9] Let A and B be bags (or sets). If the number of occurrences of each element a in A is less than or equal to the number of occurrences of a in B, then A is called a *subbag* of B, also denoted $A \subseteq B$.

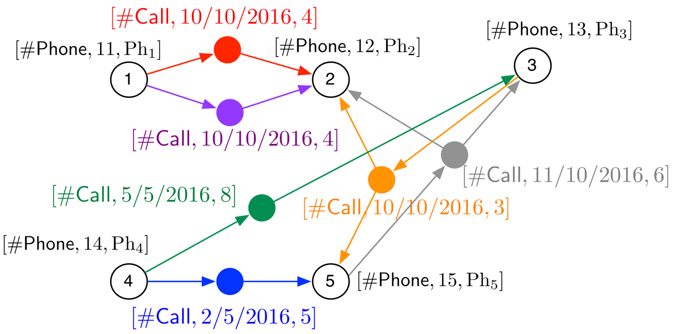


Fig. 3. Basic phone call data as a base graph.

- τ_E is a function from E to $\mathcal{E}$ (that associates a unique type to each edge of G); and
- λ_E is a function that maps a hyperedge $e \in E$ to a string $[\#\mathrm{e}, b_1, \ldots, b_{ar(\#\mathrm{n})}]$, where $\#\mathrm{e} = \tau_E(e)$ and, if $dim(\#\mathrm{e}) = (B_1, \ldots, B_{ar(\#\mathrm{n})})$, then, for $i = 1, \ldots, ar(\#\mathrm{e})$, $b_i \in dom(D_j.\ell_j)$, if B_i is the dimension D_j; λ_E is called the *edge labelling function.* □

The basic graph data that serves as input data to the graph OLAP process, is called the *base graph*. A base graph plays the role of a multi-dimensional cube in classical OLAP and is designed to contain all the information of the application domain, at the lowest level of granularity.

Definition 4 (Base graph)**.** Let dimensions $D_1, \ldots, D_d$ be given with their respective schemas and instances. The $(D_1.\text{Bottom}, \ldots, D_d.\text{Bottom})$-graphoid is called the *base graph.* □

Example 3. The running example used in this paper is aimed at analysing calls between customers of phone lines; lines correspond to different operators. Examples 1 and 2 showed some of the dimensions used as background information. Next, the call information is shown, represented as a graph. The Phone dimension plays the roles of the calling line and the callee lines (this is called a role-playing dimension in the OLAP literature [26]). The information in the hyperedges reflects the total duration of the calls between two or more phone numbers on a given day. Figure 3 shows an example of a base graph, where $N = \{1, 2, 3, 4, 5\}$ is the node set. The nodes in this base graph are all of the same type and represent phones (not persons–a person may have more than one phone). In this example, $\mathcal{N} = \{\#\mathsf{Phone}\}$. The node type $\#\mathsf{Phone}$ has arity 2. Its first attribute is a node identifier and the second one is a dimensional attribute that represents the phone number, with domain $\{\mathrm{Ph}_1, \mathrm{Ph}_2, \ldots\}$. In the example of Fig. 3,

$$\lambda_N : i \mapsto [\#\mathsf{Phone}, 10 + i, \mathrm{Ph}_i], \text{ for } i = 1, \ldots, 5.$$

Hyperedges represent phone calls, which most of the time involve two phones, but which may also involve multiple phones, representing so-called “group calls.” So, edges are all of the same type $\#\mathsf{Call}$ and $\mathcal{E} = \{\#\mathsf{Call}\}$. In Fig. 3, a directed hyperedge from a subset S of N to a subset T of N is graphically represented by a coloured node which has incoming arrows (of the same colour) from all elements of S and outgoing arrows (again of the same colour) to all elements of T. Such a coloured construction is a depiction of the hyperedge $e = (S, T)$, which will be denoted $S \rightarrow T$ from now on.[10] For example, the red and purple hyperedges $\{1\} \rightarrow \{2\}$ represent two different phone calls from Ph_1 to Ph_2, made on the same day and of the same duration. This example explains why the model assumes bags rather than

[10] The nodes of S are called the *source nodes* of e and the nodes of T are called the *target nodes* of e. The source and target nodes of e are called *adjacent* to e, and the set of the adjacent nodes to e is denoted by $Adj(e)$. Thus, $Adj(e) = S \cup T$.

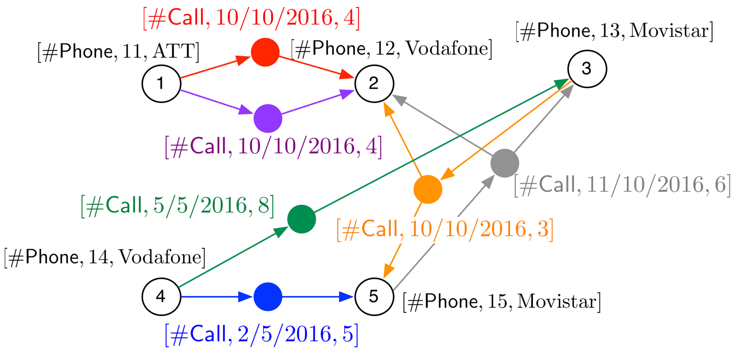


Fig. 4. A (Time.Day, Phone.Operator)-graphoid, based on the data shown in Fig. 3.

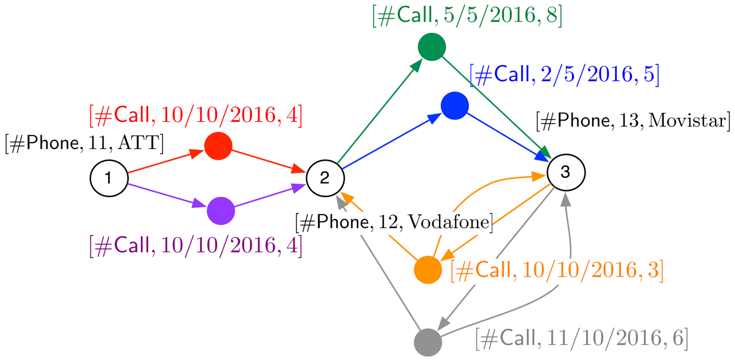


Fig. 5. An alternative (Time.Day, Phone.Operator)-graphoid, based on the data shown in Fig. 3.

sets. The orange hyperedge $\{3\} \rightarrow \{2, 5\}$ represents a group call, from Ph$_3$ to both Ph$_2$ and Ph$_5$. There are six phone calls shown in the figure. So, E is the bag $\{\!\{\{1\} \rightarrow \{2\}, \{1\} \rightarrow \{2\}, \{4\} \rightarrow \{3\}, \{4\} \rightarrow \{5\}, \{3\} \rightarrow \{2, 5\}, \{5\} \rightarrow \{2, 3\}\}\!\}$. The edge labelling function λ_E associates two attributes, with edges of type $\#\mathsf{Call}$, namely Date and Duration. Date is a dimensional attribute to which the dimensional hierarchy in Fig. 1 is associated. Duration is a measure attribute (which has as an associated aggregation function, in this case, the summation). □

Note that, although the base graph plays the role of a multi-dimensional cube in classical OLAP (or a fact table in relational OLAP), a key difference is that this cube has a variable number of "axes", since it can represent facts including a variable number of dimensions. The next example discusses two graphoids whose dimensions are at different levels of granularity. Later it will be explained how these graphoids can be obtained from the base one.

Example 4. Continuing with Example 3, consider two available dimensions, namely D_1 = Time and D_2 = Phone. A (Time.Day, Phone.Operator)-graphoid based on the base graph of Fig. 3, is shown in Fig. 4. Here, in the Phone nodes, the phone numbers have been replaced with their corresponding operator name, at the Phone.Operator level in the dimension Phone (e.g., for Ph$_3$, the corresponding operator is Movistar).

Figure 5 shows an alternative (Time.Day, Phone.Operator)-graphoid for the data from Fig. 3. This graphoid has $N = \{1, 2, 3\}$ as a node set. The nodes with identifiers 12 and 14 represent, respectively, Ph$_2$ and Ph$_4$ in the base graph (and also in the graphoid of Fig. 4), which belong to the operator Vodafone.

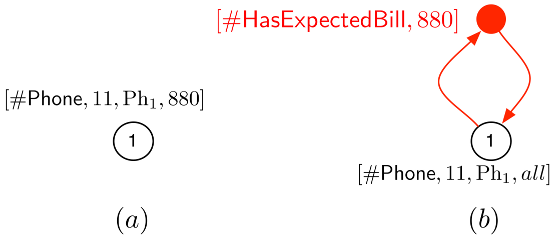


Fig. 6. (a) A node with label $[\#\mathsf{Phone}, 11, \mathrm{Ph}_1, 880]$, where 880 expresses the expected bill. (b) An edgification of this node, where the expected billing information is moved to an edge that is labelled #HasExpectedBill.

Thus, these two nodes were collapsed into one (with identifier 12) and similarly, the nodes Ph$_3$ and Ph$_5$ were collapsed into one node (with identifier 13). These operations were possible because these nodes have identical attribute values (apart from the identifier). For the dimension Time, all information in Fig. 5 is at the level of Day and all information for the dimension Phone is at the level of Company. These examples show that there can be more than one $(\mathrm{Time.Day}, \mathrm{Phone.Operator})$-graphoids "consistent" with the given base graph. Thus, some kind of normalization is needed. This is studied in the next section. □

Remark 3. Nodes are assumed to represent basic objects in the modelled application world. These objects are given by a number of descriptive attributes. Measure information, typically present in an OLAP setting to quantify facts, is, in this philosophy, represented as attributes on the hyperedges. The call duration is an example of a measure that is placed on edges of the type Call. However, the above definition also allows for node attributes to be dimensions that contain measure information. Consider a slightly modified situation in which an object of type #Phone includes an additional attribute that expresses the average (or expected) billing amount for that particular phone number, for example, $[\#\mathsf{Phone}, 11, \mathrm{Ph}_1, 880]$. In this modified setting, a user may want to compute the average expected billing amount over all phone lines. To answer these kinds of queries, attribute values of certain types of nodes must be averaged (in the example, the #HasExpectedBill attribute). However, in the model presented here, *aggregations are only performed on attribute values of hyperedges*. Whenever this problem occurs, the representation can be modified as illustrated in Fig. 6. On the left-hand side, there is a node that includes the #HasExpectedBill attribute. On the right-hand side, this attribute is brought to the *All* level in its dimension and gets the value *all*. The expected billing information is moved to a new edge of type #HasExpectedBill, where it can be subject to aggregation. The above operation is called the *edgification* of an attribute A in a node of type #n, and it is denoted by $\mathsf{Edgify}(\#\mathsf{n}, A)$. □

3.3. Minimal graphoids

In this section, the notion of *minimal* $(D_1.\ell_1, \dots, D_d.\ell_d)$-graphoid is defined. This graphoid is obtained collapsing the nodes that have identical labels (apart from the identifier) in the original graphoid. Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a $(D_1.\ell_1, \dots, D_d.\ell_d)$-graphoid. If the nodes $n_1, n_2 \in N$ have identical labels, apart from the identifier, denoted $\lambda_N(n_1) =_{\mathrm{Id}} \lambda_N(n_2)$, then these nodes are identified, such that only the one with the smallest identifier is preserved, while the others are deleted. So, if the λ_N-values of the nodes $n_1, n_2, \dots, n_k$ pairwise satisfy the $=_{\mathrm{Id}}$-relationship, and n_1 has the smallest identifier among them, then the nodes $n_2, \dots, n_k$ are replaced by n_1 and then deleted. The expression $\mathit{rep}_N(n_i) = n_1$, for $i = 1, 2, \dots, k$, indicates that n_1 represents the nodes $n_1, n_2, \dots, n_k$ in the minimal graph. All edges leaving from or arriving at the nodes $n_2, \dots, n_k$ are redirected to

n_1. For this purpose, the function rep_N is defined on subsets of the node set N: if $S \subseteq N$, then $rep_N(S) = \{rep_N(n) \mid n \in S\}$. Now, the notion of minimal graphoid is defined more formally.

Definition 5 (Minimal graphoid)**.** Let $D_0, D_1, \ldots, D_d$ and $\ell_1, \ldots, \ell_d$ be the same as in Definition 3. Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid. The *minimal graphoid of* G is the $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid $G' = (N', \tau_{N'}, \lambda_{N'}, E', \tau_{E'}, \lambda_{E'})$, defined as follows:

- N' is the set $rep_N(N) = \{rep_N(n) \mid n \in N\}$;
- $\tau_{N'}$ is a function from N' to $\mathcal{N}$, defined as $\tau_{N'}(rep_N(n)) := \tau_N(rep_N(n))$, for each n in N;
- $\lambda_{N'}$ is a function on N' defined as $\lambda_{N'}(rep_N(n)) := \lambda_N(rep_N(n))$, for each n in N;
- E' is a subbag of the set $\mathcal{P}(N') \times \mathcal{P}(N')$, defined as follows: for each hyperedge $e = S \to T$ in E, then a new hyperedge $rep_N(e) := rep_N(S) \to rep_N(T)$ is in E';
- $\tau_{E'}$ is a function from E' to $\mathcal{E}$, defined as $\tau_{E'}(rep_N(e)) := \tau_E(e)$, for each e in E;
- $\lambda_{E'}$ is a function on E' and it is defined as $\lambda_{E'}(rep_N(e)) := \lambda_E(e)$, for each e in E. □

Remark 4. The set N of nodes of G is contracted to the set $N' = rep_N(N)$, therefore each node in N' has the smallest identifier among all nodes that are mapped to n by the rep_N-function. For edges, E' is defined as the bag $\{\!\{rep_N(e) \mid e \in E\}\!\}$, which means that for each hyperedge in E, there is a corresponding hyperedge in E'. This means that the cardinalities of the bags E and E' are the same. □

Proposition 1 immediately follows from Definition 5.

Proposition 1. For any $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$, its minimal $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid always exists and it is unique. □

Example 5. The two (Time.Day, Phone.Operator)-graphoids shown in Figs 4 and 5 in Example 4, correspond to the graph of Fig. 3. The graphoid of Fig. 5 is the minimal graphoid of Fig. 4. In this example, the original nodes 2 and 4 are contracted into one node, namely the node 2 (since it has the smallest identifier of the two). Similarly, the original nodes 3 and 5 are contracted into the node 3. The original node 1 remains unchanged. Between nodes 1 and 2, there are two edges (with the same label) in the original graph. They are copied in the minimal graph. The edges between nodes 4 and 3, and 4 and 5, respectively, become two edges between the nodes 2 and 3 in the minimal graph. The two hyperedges that involve nodes 2, 3 and 5 correspond to two hyperedges between the nodes 2 and 3 in the minimal graph. □

For any $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$, the result of the minimisation described in this section is denoted $\mathsf{Minimize}(G)$, and called the *minimisation* of G.

Remark 5. It is easy to see that the minimal $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid of a $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ can be computed, in the worst case, in time that is quadratic in $|N|$ and linear in $|E|$. This can be improved, for instance, with an early pruning of the nodes that will not be contracted. Addressing this issue is beyond the scope of this paper. □

4. OLAP operations on graphs

In this section, the operations that compose the graph-OLAP language over graphoids are defined. Section 5 will show that these operations can simulate the typical OLAP operations on cubes.

4.1. Climb

The Climb-operation, intuitively, allows to define graphs at different levels of granularity, based on the background dimensions.

Definition 6 (Climb)**.** Assume a $(D_1.\ell_1, \dots, D_d.\ell_d)$-graphoid G is given as follows: $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$. Let D_k be a dimension that appears in G, and ℓ_k and ℓ'_k be levels in the schema $\sigma(D_k)$ of this dimension, such that $\ell_k \to \ell'_k$. Also, let $\rho_{\ell_k \to \ell'_k}$ be the corresponding rollup function (at the instance level). Finally, let $\#\mathsf{n}$ be a node type that appears in G, and $\#\mathsf{e}$ be an edge type that appears in G.

The *node-climb-operation of G along the dimension D_k from level ℓ_k to level ℓ'_k in all nodes of type* $\#\mathsf{n}$, denoted $\mathsf{Climb}(G, \#\mathsf{n}, D_k.(\ell_k \to \ell'_k))$, replaces all attribute values a from *dom*$(D_k.\ell_k)$ by the value $\rho_{\ell_k \to \ell'_k}(a)$ from *dom*$(D_k.\ell'_k)$, in all nodes of G of type $\#\mathsf{n}$, leaving G unaltered otherwise.

The *edge-climb-operation of G along the dimension D_k from level ℓ_k to level ℓ'_k in all hyperedges of type* $\#\mathsf{e}$, denoted $\mathsf{Climb}(G, \#\mathsf{e}, D_k.(\ell_k \to \ell'_k))$, replaces all attribute values a from *dom*$(D_k.\ell_k)$ by the value $\rho_{\ell_k \to \ell'_k}(a)$ from *dom*$(D_k.\ell'_k)$, in all edges of G of type $\#\mathsf{e}$, leaving G unaltered otherwise. □

Example 6. Applying to the graphoid G depicted in Fig. 3 the operation $\mathsf{Climb}(G, \#\mathsf{Phone}, \text{Phone}.(\text{Phone} \to \text{Operator}))$, results in the graphoid shown in Fig. 4. □

Remark 6. If a dimension D_k appears in multiple node types and edge types, to apply the Climb-operation on many of them, the shorthand expression $\mathsf{Climb}(G, \{\#\mathsf{n}_1, \dots, \#\mathsf{n}_r, \#\mathsf{e}_1, \dots, \#\mathsf{e}_s\}, D_k.(\ell_k \to \ell'_k))$ can be used. Finally, $\mathsf{Climb}(G, *, D_k.(\ell_k \to \ell'_k))$ denotes a climbing, in the dimension D_k, from level ℓ_k to level ℓ'_k in all possible node and edge types. □

4.2. Grouping

The Group-operation, both on nodes and on edges, is defined in this section.

Definition 7 (Grouping)**.** Assume a $(D_1.\ell_1, \dots, D_d.\ell_d)$-graphoid G is given as follows: $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$. Let D_k be a dimension that appears in G and let ℓ_k and ℓ'_k be levels in the schema $\sigma(D_k)$ of this dimension, such that $\ell_k \to \ell'_k$. Let $\rho_{\ell_k \to \ell'_k}$ be the corresponding rollup function. Let $\#\mathsf{n}$ be a node type that appears in G and let $\#\mathsf{e}$ be an edge type that appears in G.

The *node-grouping of G along the dimension D_k from level ℓ_k to level ℓ'_k in all nodes of type* $\#\mathsf{n}$, denoted $\mathsf{Group}(G, \#\mathsf{n}, D_k.(\ell_k \to \ell'_k))$, is defined as $\mathsf{Minimize}(\mathsf{Climb}(G, \#\mathsf{n}, D_k.(\ell_k \to \ell'_k)))$.

The *edge-grouping of G along the dimension D_k from level ℓ_k to level ℓ'_k in all hyperedges of type* $\#\mathsf{e}$, denoted $\mathsf{Group}(G, \#\mathsf{e}, D_k.(\ell_k \to \ell'_k))$, is defined as $\mathsf{Climb}(G, \#\mathsf{n}, D_k.(\ell_k \to \ell'_k))$. □

Example 7. Applying to the graphoid G depicted in Fig. 4 the operation $\mathsf{Group}(G, \#\mathsf{Phone}, \text{Phone}.(\text{Phone} \to \text{Operator}))$, results in the graphoid, depicted in Fig. 5. □

4.3. Aggregate

In this section, the Aggr-operation on measures stored in edges is defined.

Definition 8 (Aggregate)**.** Given a minimal $(D_1.\ell_1, \dots, D_d.\ell_d)$-graphoid G defined as $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$, let D_k be a dimension that appears in the hyperedges of G of type $\#\mathsf{e}$, that plays the role of a measure, to which the aggregate function F_k can be applied. The *aggregation of the graphoid G over the dimension D_k (using the function F_k)*, denoted $\mathsf{Aggr}(G, \#\mathsf{e}, D_k, F_k)$, results in a graphoid G' over the same N, τ_N and λ_N as G, with the following modified hyperedge bag E'. If the hyperedges

$e_1, e_2, \ldots, e_r$ are all of type $\#\mathsf{e}$ and all of type $S \rightarrow T$ (and if they are the only ones), and if λ_E agrees on all of them apart from a possible identifier-attribute, and apart from the dimension D_k, then the hyperedges $e_1, e_2, \ldots, e_r$ are replaced by one of them (say e_1) of the same type and with the same attribute values, apart from the identifier, which is the identifier of e_1, and the value of the attribute $D_k.\ell_k$, which becomes the value of the aggregation function F_k applied to the values of the attribute $D_k.\ell_k$ of the edges $e_1, e_2, \ldots, e_r$. □

Example 8. Applying the operation $\mathsf{Aggr}(G, \#\mathsf{Call}, \text{Duration}, \text{SUM})$ to the graphoid G, depicted in Fig. 5, results in a graphoid where the two edges that connect the nodes 1 and 2 are replaced by one edge with label $[\#\mathsf{Call}, 10/10/2016, 8]$, which contains, in the measure attribute, the sum of the two durations. □

Remark 7. To aggregate multiple dimensions $M_1, \ldots, M_k$, using the aggregate functions $F_1, \ldots, F_k$ simultaneously, the notation would be: $\mathsf{Aggr}(G, \#\mathsf{e}, \{M_1, \ldots, M_k\}, \{F_1, \ldots, F_k\})$. Also, for simplicity, only the typical SQL aggregate functions SUM, MAX,MIN and COUNT are considered. □

Remark 8. Although the operations Climb, Group, and Aggr, are not present in classic relational OLAP, they are included here for several reasons: first, they can be useful when operating on graphs in practice; second, they facilitate and make it simple the definition of the Roll-up operation, that otherwise could be unnecessarily difficult to express. □

4.4. Roll-Up

The operations defined above allow defining the Roll-Up-operation over dimensions and measures stored in edges, as explained next.

Definition 9 (Roll-Up)**.** Assume a $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid G is given as follows: $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$. Let D_c be a dimension that appears in some nodes and/or hyperedges of G, that plays the role of a climbing dimension. Let $M_1, \ldots, M_k$ be dimensions that appear in the hyperedges of type $\#\mathsf{e}$ of G. These dimensions play the role of measure dimensions, and it is assumed that aggregate functions $F_1, \ldots, F_k$ are associated with them. Let $\#\mathsf{n}_1, \ldots, \#\mathsf{n}_r$ be node types appearing in G, and let $\#\mathsf{e}_1, \ldots, \#\mathsf{e}_s$ be hyperedge types appearing in G. The *roll-up of G over the dimensions $M_1, \ldots, M_k$ (using the functions $F_1, \ldots, F_k$) in hyperedges of type $\#\mathsf{e}$, and over the climbing dimension D_c from level ℓ_c to level ℓ'_c in nodes of types $\#\mathsf{n}_1, \ldots, \#\mathsf{n}_r$ and edges of types $\#\mathsf{e}_1, \ldots, \#\mathsf{e}_s$,* denoted

$$\mathsf{Roll\text{-}Up}(G, \{\#\mathsf{n}_1, \ldots, \#\mathsf{n}_r, \#\mathsf{e}_1, \ldots, \#\mathsf{e}_s\}, D_c.(\ell_c \rightarrow \ell'_c); \#\mathsf{e}, M_1, \ldots, M_k, F_1, \ldots, F_k),$$

is defined as

$$\mathsf{Aggr}(\mathsf{Minimize}(\mathsf{Climb}(G, \{\#\mathsf{n}_1, \ldots, \#\mathsf{n}_r, \#\mathsf{e}_1, \ldots, \#\mathsf{e}_s\},$$
$$D_c.(\ell_c \rightarrow \ell'_c))), \#\mathsf{e}, M_1, \ldots, M_k, F_1, \ldots, F_k).$$

□

Example 9. Applying to the graphoid depicted in Fig. 5 the operation $\mathsf{Roll\text{-}Up}(G, \{\#\mathsf{Call}\}, \text{Time}.(\text{Day} \rightarrow \text{Year}); \#\mathsf{Call}, \text{Duration}, \text{SUM})$, results in the graphoid of Fig. 7. The minimisation step in the above implementation of the roll-up operation does nothing, in this case, since the operation is applied to a minimal graphoid. □

Remark 9. To apply the climbing in the roll-up operation to the nodes and edges of all possible types, the shorthand "$*$" is used as follows: $\mathsf{Roll\text{-}Up}(G, *, D_c.(\ell_c \rightarrow \ell'_c); \#\mathsf{e}, M_1, \ldots, M_k, F_1, \ldots, F_k)$. To aggregate over all edge types, the notation is $\mathsf{Roll\text{-}Up}(G, *, D_c.(\ell_c \rightarrow \ell'_c); *, M_1, \ldots, M_k, F_1, \ldots, F_k)$. □

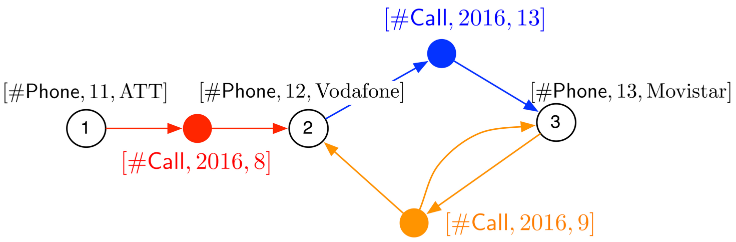


Fig. 7. The result of the operation $\mathsf{Roll\text{-}Up}(G, \{\#\mathsf{Phone}\}, \mathrm{Time}.(\mathrm{Day} \to \mathrm{Year}); \#\mathsf{Call}, \mathrm{Duration}, \textsc{Sum})$ applied to the graphoid of Fig. 5.

4.5. Drill-Down

The Drill-Down-operation does the opposite of Roll-Up,[11] taking a graphoid to a finer granularity level, along a dimension D_d, call it a *descending* dimension, and also operating over a collection of measures, using the same aggregate functions associated with such measures. Note also that, descending from a level ℓ_d down to a level ℓ'_d along a dimension D_d is equivalent to climbing from the bottom level of D_d, D_d.Bottom, to the level ℓ'_d along D_d. Thus, the *drill-down of G over the dimensions $M_1, \ldots, M_k$ (using the functions $F_1, \ldots, F_k$) in hyperedges of type $\#\mathsf{e}$, and over the descending dimension D_d from level ℓ_d to level ℓ'_d in nodes of types $\#\mathsf{n}_1, \ldots, \#\mathsf{n}_r$ and edges of types $\#\mathsf{e}_1, \ldots, \#\mathsf{e}_s$*, denoted

$$\mathsf{Drill\text{-}Down}(G, \{\#\mathsf{n}_1, \ldots, \#\mathsf{n}_r, \#\mathsf{e}_1, \ldots, \#\mathsf{e}_s\},$$
$$D_d.(\ell_d \to \ell'_d); \#\mathsf{e}, M_1, \ldots, M_k, F_1, \ldots, F_k),$$

is defined as

$$\mathsf{Aggr}(\mathsf{Minimize}(\mathsf{Climb}(G, \{\#\mathsf{n}_1, \ldots, \#\mathsf{n}_r, \#\mathsf{e}_1, \ldots, \#\mathsf{e}_s\},$$
$$D_d.(\mathrm{Bottom} \to \ell'_d))), \#\mathsf{e}, M_1, \ldots, M_k, F_1, \ldots, F_k).$$

Given the above, in what follows the discussion is limited to the Roll-Up-operation.

4.6. Dice

The Dice-operation over a graphoid, produces a subgraphoid that satisfies a Boolean condition φ over the available dimension levels. A "strong" version is also defined, called the s-Dice-operation. In this context, φ is a Boolean combination of atomic conditions of the form $D.\ell < c$, $D.\ell = c$, and $D.\ell > c$, where D is a dimension, ℓ is a level in that dimension, and $c \in \mathit{dom}(D.\ell)$. The expression φ can be written in disjunctive normal form as

$$\bigvee_k \bigwedge_l \varphi_{kl},$$

where all φ_{kl} are atomic conditions.

Before giving the definition of the Dice-operation, it must be explained what does it mean that a hyperedge e in a graphoid *satisfies* φ, denoted $e \models \varphi$. For this, interpreting conjunction and disjunction

[11] Actually, this is true for a sequence of roll-up and drill-down operations such that there are no slicing or dicing operations (explained in Sections 4.6 and 4.7) in-between. However, for the sake of simplicity, and without loss of generality, in this paper it is assumed that roll-up and drill-down are the inverse of each other.

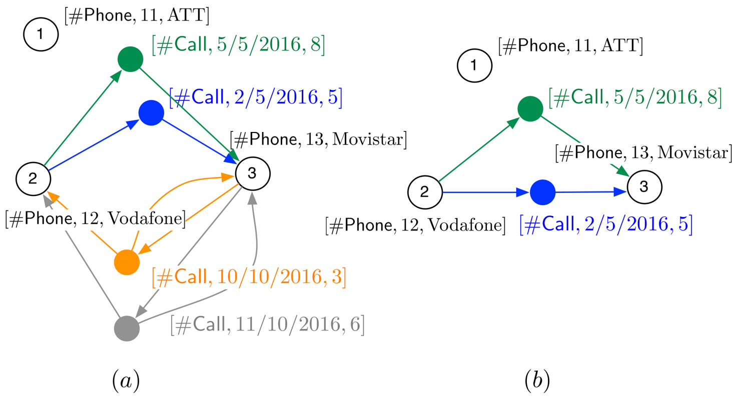


Fig. 8. (a) The result of the operation $\mathsf{Dice}(G, \text{Phone.Operator} \neq \text{ATT})$ to the graphoid depicted in Fig. 5; (b) The result of applying $\mathsf{Dice}(G, \text{Time.Month} = 5/2016)$ to (a).

in the usual way, it suffices to define $e \models \varphi_{kl}$ for the atomic formulas that appear in φ. Thus, φ_{kl} cannot be evaluated in e if the label of e does not contain information on dimension D at level ℓ. Otherwise, φ_{kl} can be evaluated in e. Let φ_{kl} be $D.\ell < c$, $D.\ell = c$ or $D.\ell > c$; φ_{kl} is *not false* in e if it can be evaluated in e and is true, or if it cannot be evaluated in e. The notion of φ_{kl} being *not false* in a node n adjacent to e (that is, $n \in \mathit{Adj}(e)$) is defined analogously. Finally, $e \models \varphi_{kl}$ if φ_{kl} is not false in e and not false in all $n \in \mathit{Adj}(e)$.

Definition 10 (Dice)**.** Assume a $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid G is given as $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$. Let φ be a Boolean combination of equality and inequality constraints that involve, on the one hand dimension levels $\ell'_1, \ldots, \ell'_d$ (equal or higher than $\ell_1, \ldots, \ell_d$ in the dimension schemas $\sigma(D_1), \ldots, \sigma(D_d)$, respectively), and on the other hand, constants from $\mathit{dom}(D_1.\ell'_1), \ldots, \mathit{dom}(D_d.\ell'_d)$. The *dice over* G *on the condition* φ, denoted $\mathsf{Dice}(G, \varphi)$, produces a subgraphoid of G, whose nodes are the nodes of G and whose edges satisfy the conditions expressed by φ. When an hyperedge does not satisfy φ, the whole hyperedge is deleted from the graph and thus, it does not belong to $\mathsf{Dice}(G, \varphi)$. All other edges of G belong to $\mathsf{Dice}(G, \varphi)$. If two edges in G have the same set of adjacent nodes and one of them is deleted from G in $\mathsf{Dice}(G, \varphi)$, then both of them are deleted in G to obtain the *strong dice over* G *on the condition* φ, denoted $\mathsf{s\text{-}Dice}(G, \varphi)$. □

Example 10. Applying the operation $\mathsf{Dice}(G, \text{Phone.Operator} \neq \text{ATT})$ to the graphoid depicted in Fig. 5, results in the graphoid of Fig. 8a. In this case, the result would be the same as the one obtained after applying $\mathsf{s\text{-}Dice}(G, \text{Phone.Operator} \neq \text{ATT})$. Applying $\mathsf{Dice}(G, \text{Time.Month} = 5/2016)$ over the graphoid in Fig. 8a, results in the graphoid shown in Fig. 8b. □

4.7. Slice

Intuitively, the Slice operation eliminates the references to a dimension in a graphoid. The formal definition follows.

Definition 11 (Slice)**.** Assume a $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid G is given as $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$. Let D_s be a dimension that appears in some nodes and/or hyperedges of G. Let $M_1, \ldots, M_k$ be dimensions that appear in the hyperedges of G. These dimensions play the role of measure dimensions. It is assumed that aggregate functions $F_1, \ldots, F_k$ are associated with them. The *slice of*

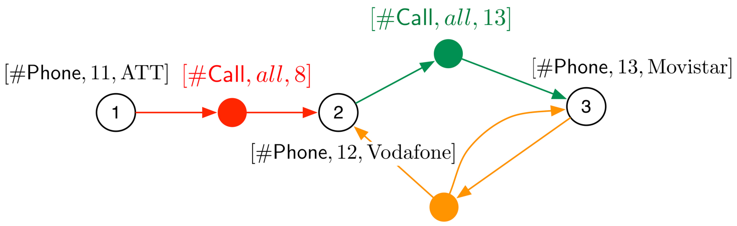


Fig. 9. The result of the operation Slice$(G$, Time, Duration, SUM$)$ on the graphoid of Fig. 5.

the dimension D_s from G over the dimensions $M_1, \ldots, M_k$ (using the functions $F_1, \ldots, F_k$), denoted Slice$(G, D_s; M_1, \ldots, M_k, F_1, \ldots, F_k)$, is defined as the roll-up operation up to the level $D_s.All$ over the dimensions $M_1, \ldots, M_k$ (using the functions $F_1, \ldots, F_k$). Formally, this slice operation is defined as Roll-Up$(G, *, D_s.(\ell_s \to All); *, M_1, \ldots, M_k, F_1, \ldots, F_k)$. □

Example 11. Applying the operation Slice$(G$, Time; Duration, SUM$)$ to the graphoid depicted in Fig. 5, results in the graphoid of Fig. 9. □

4.8. Node-delete

The n-Delete-operation over a graphoid, deletes all nodes of a certain type and delete, in the source and target set of all edges, the nodes of this type. Again, although this operation is not present in classic OLAP, it is needed to simulate the classic OLAP slice operation, as will become clear in Section 5.2.

Definition 12 (Node-delete)**.** Assume a $(D_1.\ell_1, \ldots, D_d.\ell_d)$-graphoid G is given as $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$. Given a node type #n, the *node-delete over G* operation, denoted n-Delete$(G, \#\mathsf{n})$, produces a subgraphoid of G, whose nodes of type #n are deleted, and such that all edges $e = S \to T$ are replaced by edges $S^{\#\mathsf{n}} \to T^{\#\mathsf{n}}$, where $S^{\#\mathsf{n}}$ and $T^{\#\mathsf{n}}$ are S and T, respectively, minus the nodes of type #n. The edges remain of the same type and they keep the same label. □

Example 12. When a graphoid contains only nodes of one type, as in Fig. 3, the result of the deletion of a node is, obviously, the empty graph. In the graphoid of Fig. 11 (explained later), the result of n-Delete$(G, \#\mathsf{Location})$ would be a graph with nodes 2 and 3, where a hyperedge containing only these nodes would remain, with label [#Sales, 10]. □

5. Classical OLAP cubes as a special case of OLAP graphs

This section explains how the classical cube-based OLAP model can be represented in the graphoid OLAP model. It is also shown that the classical OLAP-operations Roll-Up, Drill-Down, Slice and Dice can be simulated by the graphoid OLAP-operations defined in Section 4.

5.1. A discussion on modelling cubes as graphoids

Figure 10 illustrates a typical example of an OLAP cube with dimensions $(D_1, D_2, D_3) = (Product, Location, Time)$. The cube represents sales amounts of products at certain stores locations (cities) on certain dates (at the lowest level of granularity). There are several ways for representing this cube in the graphoid model. Figure 11 shows two ways of modelling the fact $(Lego, Antwerp, 1/1/2014; 10)$, which expresses that the sales of Lego in the Antwerp store on January 1st, 2014 amount to 10.

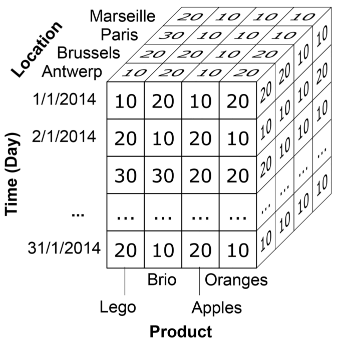


Fig. 10. An example of a Sales data cube with one measure: $\mu_1 = \textit{sales}$.

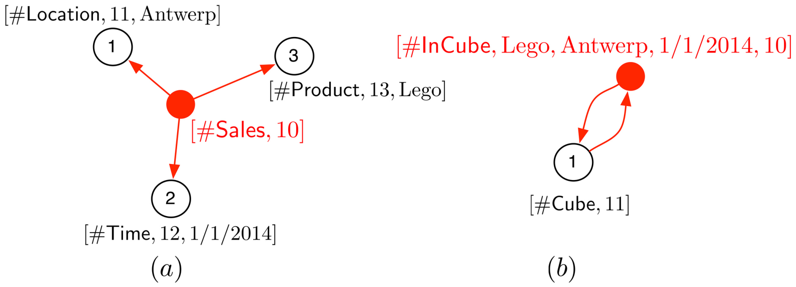


Fig. 11. Star-representation of the fact (*Lego*, *Antwerp*, 1/1/2014; 10) (a). Petal-representation of the fact (*Lego*, *Antwerp*, 1/1/2014; 10) (b).

Figure 11a shows nodes 1, 2 and 3, of types #Product, #Location and #Time, respectively. All of them have only one attribute, to store the values *Lego*, *Antwerp* and 1/1/2014, call those attributes ProductVal, LocationVal and TimeVal, respectively. Further, those attributes are dimensions, with an appropriate dimension schema. The measure information is stored in the hyperedge $\emptyset \to \{1, 2, 3\}$ with label [#Sales, 10], which has one attribute, namely SalesVal, to store the sale amount (10, in this case). Thus, in this approach, each cell of a data cube is modelled by a "star"-shaped hyperedge.

A more compact representation is shown in Fig. 11b. Here, there is only one node, of type #Cube in the graphoid, which represents the data cube. This node is labelled [#Cube, 11], and has no attribute values (apart from an identifier value). Cell-coordinates and cell-content are stored in hyperedges that form loops around the node. The fact (*Lego*, *Antwerp*, *1/1/2014*; 10) is modelled by a unique hyperedge with label [#InCube, Lego, Antwerp, 1/1/2014, 10]. Thus, cube facts are represented by a hyperedge of type #InCube that has four attributes: ProductVal, LocationVal, TimeVal and SalesVal.

In between the two alternatives above, there are, obviously, more modelling possibilities. The next section will show that the graphoid OLAP-operations presented in Section 4, are at least as powerful as the classical OLAP-operations of the classical cube model. The proof will assume the star-representation of data cubes in the graphoid model (Fig. 11a).

5.2. Graph- and classic-OLAP operations equivalence

A classical data cube is based on dimensions $D_1, \ldots, D_d$ and on measures $\mu_1, \ldots, \mu_m$. Each dimension D_i has its domain $dom(D_i)$ and its dimension schema $\sigma(D_i)$ with corresponding dimension instances and roll-up functions (between the levels). Each measure μ_j has its domain $dom(\mu_j)$

and an associated aggregation function f_j. A *(classical) data cube* C *over dimensions* $D_1, \ldots, D_d$ *with measures* $\mu_1, \ldots, \mu_m$, can then be seen as a partial function $\mu : \mathit{dom}(D_1) \times \cdots \times \mathit{dom}(D_d) \to \mathit{dom}(\mu_1) \times \cdots \times \mathit{dom}(\mu_m)$. This function maps each "cell" of the cube to m values for the measures. A cell of the cube with coordinates $(a_1, \ldots, a_d) \in \mathit{dom}(D_1) \times \cdots \times \mathit{dom}(D_d)$, that contains values $(c_1, \ldots, c_m) \in \mathit{dom}(\mu_1) \times \cdots \times \mathit{dom}(\mu_m)$, is denoted by $(a_1, \ldots, a_d; c_1, \ldots, c_m)$. Below, the "star-representation" of a data cube in the graphoid model is formally defined.

Definition 13 (Star-graphoid)**.** Let C be a data cube over dimensions $D_1, \ldots, D_d$, with measures $\mu_1, \ldots, \mu_m$. The *star-graphoid* of C, denoted $\mathsf{Star}(C)$, is defined as follows.

- For $i = 1, \ldots, d$, for each $a_i \in \mathit{dom}(D_i)$, there is a node of type $\#D_i$ with label $[\#D_i, id, a_i]$, where id is a unique node identifier.
- For each cell $(a_1, \ldots, a_d; c_1, \ldots, c_m) \in \mathit{dom}(D_1) \times \cdots \times \mathit{dom}(D_d) \to \mathit{dom}(\mu_1) \times \cdots \times \mathit{dom}(\mu_m)$, there are m hyperedges: for each $j = 1, \ldots, m$, there is a hyperedge of type $\#\mu_j$ with an empty source node set and with a target node set consisting of all nodes labelled $[\#D_i, id, a_i]$, for $i = 1, \ldots, d$, which is labelled $[\#\mu_j, c_j]$. □

Now, the main theorem of this section is stated.

Theorem 1. The cube OLAP-operations $\mathsf{Roll\text{-}Up}$, $\mathsf{Drill\text{-}Down}$, Slice and Dice can be expressed (or simulated) by OLAP-operations on graphoids.

Proof. Let C be a data cube, and let $\mathsf{Star}(C)$ be its star-graphoid. The proof is based on showing that each of the classical OLAP operations $\mathsf{Roll\text{-}Up}$, $\mathsf{Drill\text{-}Down}$, Slice and Dice, over C, can be equivalently applied on $\mathsf{Star}(C)$. The semantics for the classical OLAP operations is the one given in [17].

Roll-Up. For cube data, a roll-up operation takes as input a data cube C, a dimension D_c and a level ℓ_i in $\sigma(D_c)$ and returns the aggregation of the original cube along D_c up to level ℓ_c for all of the input measures $\mu_1, \ldots, \mu_m$, using aggregate functions $F_1, \ldots, F_m$. Assume, without loss of generality, that the roll-up starts at the *Bottom* level, that is, at $\mathit{dom}(D_c)$. Also assume, for the sake of clarity of exposition, that $m = 1$, that is, that there is only one measure, call it μ, with associated aggregate function F. Now, it will be shown that the roll-up $\mathsf{Roll\text{-}Up}(C, D_c.\ell_c; \mu, F)$ on the cube C can be simulated on $\mathsf{Star}(C)$ by the graphoid OLAP-operation $\mathsf{Roll\text{-}Up}(\mathsf{Star}(C), \{\#\mathrm{D}_c\}, D_c.(\mathit{Bottom} \to \ell_c); \#\mathsf{e}_\mu; \mu, F)$, where $\#\mathrm{D}_c$ is the unique node type in $\mathsf{Star}(C)$ that contains information on D_c and where $\#\mathsf{e}_\mu$ is the unique edge type that contains measure information on μ.

Let $(a_1, \ldots, a_{c-1}, a_{c+1}, \ldots, a_d)$ be an element of $\mathit{dom}(D_1) \times \cdots \mathit{dom}(D_{c-1}) \times \mathit{dom}(D_{c+1}) \times \cdots \times \mathit{dom}(D_d)$ and suppose that there are r values $a_{c,i}$ from $\mathit{dom}(D_c)$ (for $i = 1, \ldots, r$) such that $(a_1, \ldots, a_{c-1}, a_{c,i}, a_{c+1}, \ldots, a_d; m_i)$ appear in the cube C, and such that all $a_{c,i}$ roll-up to the same element, call it a_{ru}, that means $\rho_{D_c.\mathit{Bottom}_k \to \ell_c}(a) = a_{ru}$. The roll-up on C will replace these r cells by one "new" cell which has coordinates $(a_1, \ldots, a_{c-1}, a_{ru}, a_{c+1}, \ldots, a_d)$ in $\mathit{dom}(D_1) \times \cdots \mathit{dom}(D_{c-1}) \times \mathit{dom}(D_c.\ell_c) \times \mathit{dom}(D_{c+1}) \times \cdots \times \mathit{dom}(D_d)$, and which contains the aggregated measure $F(\{m_1, \ldots, m_r\})$. In $\mathsf{Star}(C)$, each one of these "new" cells will be represented by a hyperedge. To achieve this, the following graphoid OLAP-operation is performed:

$$\mathsf{Roll\text{-}Up}(\mathsf{Star}(C), \#\mathrm{D}_c, D_c.(\mathit{Bottom} \to \ell_c); \#\mathsf{e}_\mu, \mu, F).$$

To see the correctness of this claim, the substeps in the above graphoid roll-up are analysed. First, $\mathsf{Climb}(\mathsf{Star}(C), \#\mathrm{D}_c, D_c.(\mathit{Bottom} \to \ell_c))$ is performed; a graphoid called G_1 is obtained. Compared against $\mathsf{Star}(C)$, in G_1 all nodes and edges remain the same, except for the nodes of type $\#\mathrm{D}_c$, which now contain values at level ℓ_c. Next, a minimisation is performed (to obtain a grouping on

D_c), which may contract some nodes in G_1 into "roll-up" nodes. Call the resulting graphoid G_2. These roll-up nodes of G_2 simulate the "new" cells in the cube that store the aggregate information. Finally, $\mathsf{Aggr}(G_2, \#\mathrm{e}_\mu, \mu, F)$ contracts edges that have the same adjacency set and gives them the aggregated value of μ as attribute value.

Drill-Down. As mentioned above, the drill-down to level ℓ can be seen as a roll-up from the *Bottom* level to level ℓ. Therefore, no proof is needed.

Slice. On data cubes, the Slice-operation takes as input a cube C, a dimension D_s and returns a cube in which the dimension D_s is dropped, and all measures are aggregated over the dropped dimension. To drop the dimension D_s, a roll-up to the level *All* in this dimension is needed first, such that its domain becomes a singleton. Thus, to simulate this on $\mathsf{Star}(C)$ using graphoid OLAP-operations, a climb to the level *All* in the dimension D_s is performed, and therefore the proof of the roll-up case holds, taking into account that all nodes representing D_s will contain the value "*all*". Thus, the slice of the cube C is simulated by $\mathsf{Slice}(\mathsf{Star}(C), D_s; \mu, F)$. There one step missing, however. When slicing a dimension from a cube C, this dimension is deleted. In the case of the graphoid $\mathsf{Star}(C)$, the nodes of type $\#D_s$ are still present in $G_1 = \mathsf{Slice}(\mathsf{Star}(C), D_s; \mu, F)$. So, $\mathsf{n\text{-}Delete}(G_1, \#\mathrm{D}_s)$ is needed to delete these nodes.

Dice. Intuitively, the $\mathsf{Dice}(C, \varphi)$ operation, where φ is a Boolean condition over level values and measures, selects the cells in a cube C that satisfy φ. The resulting cube has the same dimensionality as the original cube. It must be shown that $\mathsf{Dice}(C, \varphi)$ can be simulated by $\mathsf{s\text{-}Dice}(\mathsf{Star}(C), \varphi)$. As in Section 4.6, take

$$\varphi = \bigvee_k \bigwedge_l \varphi_{kl},$$

with φ_{kl} of the form $D.\ell < c$, $D.\ell = c$ or $D.\ell > c$, where D is a dimension, ℓ is a level in that dimension and $c \in \mathit{dom}(D.\ell)$; or $\mu < c$, $\mu = c$ or $\mu > c$, where μ is a measure and c belongs to the domain of that measure.

Let $(a_1, \ldots, a_d; c_1, \ldots, c_m) \in \mathit{dom}(D_1) \times \cdots \times \mathit{dom}(D_d) \to \mathit{dom}(\mu_1) \times \cdots \times \mathit{dom}(\mu_m)$ be a cell of C that satisfies φ. Denote this by $(a_1, \ldots, a_d; c_1, \ldots, c_m) \models \varphi$. The proof here requires showing that the edges e_j, labelled $[\#\mu_j, c_j]$ (that are adjacent to the nodes $[\#D_i, id, a_i]$, for $i = 1, \ldots, d$), for $j = 1, \ldots, m$, also satisfy φ. From $(a_1, \ldots, a_d; c_1, \ldots, c_m) \models \varphi$ it follows that there exists a k such that for all l, $(a_1, \ldots, a_d; c_1, \ldots, c_m) \models \varphi_{kl}$ holds.

If φ_{kl} is of the form $D.\ell < c$, $D.\ell = c$ or $D.\ell > c$, then φ_{kl} is undefined in the edge label and thus, it is not false in it. Furthermore, because of the particular definition of stars in star-graphoids, where all nodes that are adjacent to an edge e_j carry information on unique dimensions, φ_{kl} is not false in all adjacent nodes that do not contain information on $D.\ell$ and it is true in the unique adjacent node that contains information on $D.\ell$. Therefore, the edge e_j satisfies φ_{kl}.

If φ_{kl} is of the form $\mu < c$, $\mu = c$ or $\mu > c$, then φ_{kl} evaluates to true on one of the edges e_j (that contains information on that measure μ) and is undefined on the other edges (that contain information on other measures). On the adjacent nodes to these edges, the condition φ_{kl} is not false (since these nodes do not contain information on any measures). In both cases, all these edges satisfy φ_{kl}. This means that the strong dice-operation will keep all these edges.

By a similar reasoning, it can be shown that when $(a_1, \ldots, a_d; c_1, \ldots, c_m) \not\models \varphi_{kl}$, $e_j \not\models \varphi_{kl}$ holds.

This shows that exactly the edges (labelled $[\#\mu_j, c_j]$) corresponding to cells $(a_1, \ldots, a_d; c_1, \ldots, c_m)$, where φ is not satisfied are deleted from the graphoid $\mathsf{Star}(C)$ by the strong dice-operation. This completes the proof. □

6. Case study and discussion

The running example followed so far in this paper will also be used as a case study, in order to evaluate the hypergraph model against the traditional relational OLAP alternative. The example case has many interesting characteristics, such as: (a) Normally it involves huge volumes of data facts (i.e., calls); (b) The number of dimensions involved in facts is variable, since calls may differ from one another in the number of participants; (c) It allows performing not only the typical OLAP operations described in Section 4, over the fact measures, but also to aggregate the graph elements using graph measures like shortest paths, centrality, and so on. Therefore, the case study is appropriate for illustrating and discussing the graphoid model usefulness in two situations: (a) The classic OLAP scenario, where the relational model is normally used; and (b) A *Graph OLAP* scenario, where graph metrics are aggregated. The hypothesis to be tested here is that, although the relational OLAP alternative works better in scenario (a), when facts have a fixed dimensionality (e.g., when all calls in the database involve the same number of participants), the graphoid model is competitive when the number of dimensions is variable, and definitely better for scenario (b), where queries compute aggregations over graph metrics.

The dataset to analyse consists of group calls between phone lines, where a line cannot call itself, and the analyst also needs to identify the line who started the call. The schemas of the background dimensions are the ones in Fig. 1, with small changes that will be explained below. Facts are similar to the ones in Fig. 3.

Although performing an exhaustive experimental study is beyond the scope of this paper, and will be part of future work, this section aims at *analysing the plausibility of the graph model* to become a better solution that the relational model for the kinds of problems where factual data are naturally represented as graphs. For this, the graphoid model are compared against the relational alternative containing exactly the same data. First, two alternative relational OLAP representations are implemented on a PostgreSQL database, and three synthetic datasets of different sizes are produced and loaded into both representations. Then, the same datasets are loaded into a graph database. Neo4j is used for this purpose, and queries are written in Cypher, Neo4j's high level query language.[12]

6.1. Relational representation

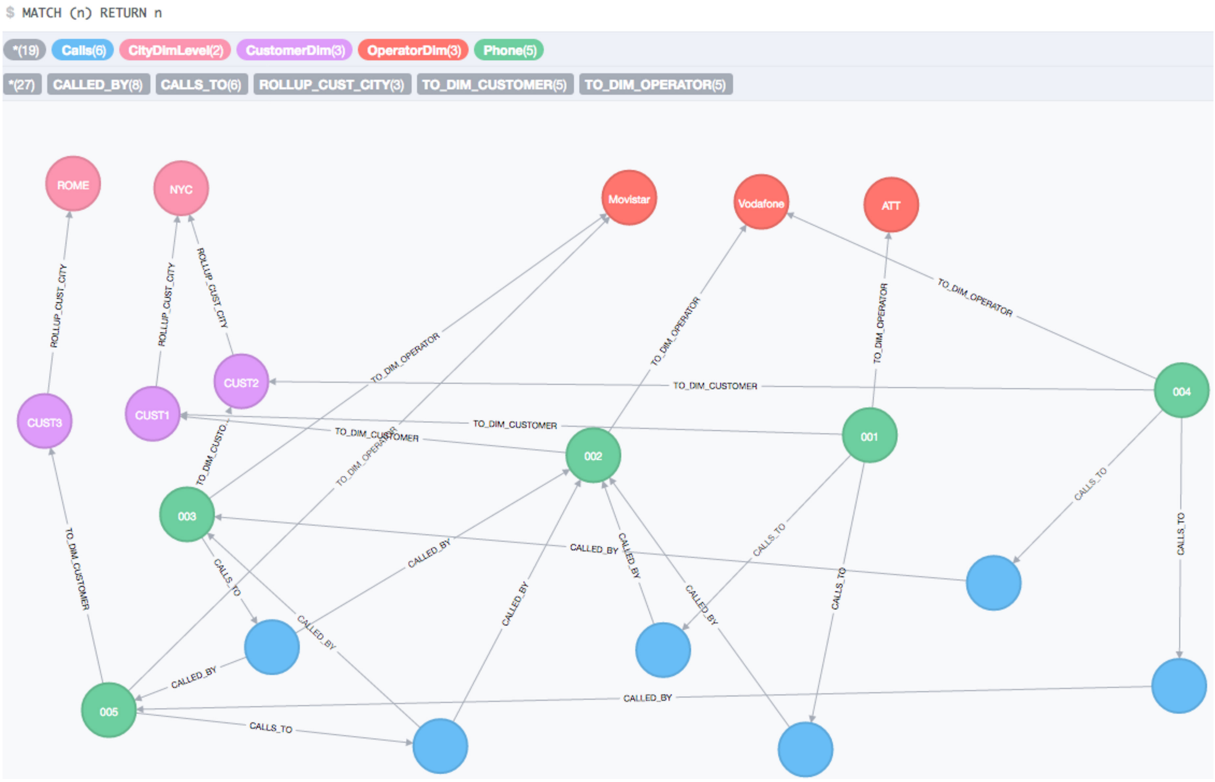
Since the relational design may impact in query performance, two alternative designs for the fact table are implemented in order to provide a fair comparison. In both cases, the fact table schema is the following: Calls(CallId, CallerId, Participant,StartTime, EndTime, Duration).

The meaning of the attributes is:

- CallId: Call identifier;
- CallerId: The identifier of the line which initiated the call;
- StartTime, EndTime: Initial and final instants of the call;
- Duration: Attribute precomputed as (StartTime – EndTime).

Although the schemas are the same in both cases, the instances differ from each other. In one case, a call between phone Ph_1, Ph_2, and Ph_3, initiated by Ph_1, contains the tuples $(\mathit{1}, Ph_1, Ph_2)$ and $(\mathit{1}, Ph_1, Ph_3)$. In the other case, a tuple $(\mathit{1}, Ph_1, Ph_1)$ is added to the other two to indicate that Ph_1 started the call. This makes a difference for queries where the user is not interested in who did initiate the call. In what follows, both relational representations are denoted Calls and Calls-alt, respectively.

[12] https://neo4j.com/developer/cypher-query-language/.



```
$ MATCH (n) RETURN n
```

Fig. 12. Portion of the call-graph.

As expressed above, the background dimensions are the same of Fig. 1. There are two slight differences, however, for practical reasons. First, for the Time dimension, the bottom level has granularity TIMESTAMP, since the StartTime and EndTime attributes in the fact tables have that granularity. That means, a new level is added to the dimension. Second, in the Phone dimension the bottom level is the phone identifier, denoted Id, which rolls up to the line number, denoted Number. This is because the caller and the callee are represented as integers, as usual in real world data warehouses. The Phone dimension is represented in a single table, keeping the constraints indicated by the hierarchies. This representation (i.e., Star) was chosen to provide a fair comparison. In summary, the dimension table schema is Phone(Id, Number, Customer, City, Country, Operator).

6.2. Graphoid-OLAP representation

The logical model for the graphoid representing the calls (i.e., the base graphoid), is similar to the one depicted in Fig. 3. There are two main entity nodes, namely #Phone and #Call, to represent call facts. These are linked through edges labelled #creator and #receiver, the former going from the phone that initiated the call, to the node representing such call. Background dimensions are represented in the same graph, using the entity nodes #Operator, #User, #City and #Country for the dimension levels. Finally, dimension levels are linked using the edges of types #provided_by, #has_phone, #belongs_to and #lives_in. It can be observed that nodes are not duplicated. Figure 12 shows a portion of the running example implemented in Neo4j.

Table 1
Dataset sizes for the relational representation

Dataset	Tuples Calls	Tuples Calls-alt	Calls	Tuples Phone
D1	293,817	420,517	126,700	793
D2	528,408	756,117	227,709	4,689

Table 2
Dataset sizes for the graph representation

Dataset	Phone nodes	User nodes	Call nodes	Creator edges	Receiver edges
D1	793	500	126,700	126,700	293,817
D2	4,689	3,000	227,710	227,709	528,408

```
set myvars.recsize = 2;  --3,4, 5 or any number
WITH RECURSIVE records(CallId, ids, Duration) as
(
SELECT CallId, array[Number], Duration
FROM Calls  JOIN Phone AS member ON
Calls.Participant = member.PhoneId

UNION

SELECT  CallId, array[Number], Duration
FROM Calls  JOIN  Phone AS member ON
Calls.CallerId = member.PhoneId

UNION

SELECT Calls.CallId, array (SELECT unnest(array[Number] || ids)
       As x ORDER BY x), Calls.Duration
FROM Calls JOIN Phone AS member ON Calls.Participant = member.Id,
records
WHERE Calls.CallId = records.CallId
AND array_length(ids, 1) < current_setting('myvars.recsize')::int
AND  Number <> ALL(ids)
)
SELECT ids, avg(duration)
FROM records
WHERE array_length(ids, 1) > 1
GROUP BY ids;
```

Fig. 13. Query 1 – Calls representation.

6.3. Datasets

For the relational representation, synthetic datasets of two different sizes are generated and loaded into a PostgreSQL database. Table 1 depicts the sizes of the datasets. The first column shows the number of tuples in the Calls fact table. The second column shows the number of tuples in the Calls-alt fact table. The third column indicates the number of calls (only one column, since the number of calls is the same in both versions), and the fourth column tells the number of tuples in the Phone dimension table.

For the graph representation, Table 2 depicts the main numbers of elements in the Neo4j graph.

6.4. Queries

This section shows how different kinds of complex analytical queries can be expressed and executed

```
MATCH  (p1:Phone)-[:creator|:receiver]-(m:Call)
       -[:creator|:receiver]-(p2:Phone)
WHERE p1.id < p2.id
RETURN  p1,p2, avg(m.duration)
```

Fig. 14. Query 1 – Cypher query for $N = 2$.

```
MATCH (t1 :Phone)<-[:creator|:receiver]-(c :Call)-
      [:creator|:receiver]->(t2 :Phone),(t3:Phone)
      <-[:creator|:receiver]-(c :Call)
WHERE t1.number < t2.number  AND t2.number < t3.number
RETURN t1.number, t2.number, t3.number, avg(c.duration)
```

Fig. 15. Query 1 – Cypher query for $N = 3$.

```
MATCH (u1:User)<-[:has_phone]-(t1:Phone)<-[:creator|:receiver]-
      (c:Call)-[:creator|:receiver]->(t2:Phone)-[:has_phone]->
      (u2:User), (u3:User)<-[:has_phone]-(t3:Phone)<-
      [:creator|:receiver]-(c:Call)
WHERE u1.id < u2.id  AND u2.id < u3.id
RETURN u1.name, u2.name, u3.name, avg(c.duration)
```

Fig. 16. Query 2 – Cypher for $N = 3$.

over the three representations described above. Four kinds of OLAP queries are discussed: (a) Queries where the aggregations are performed for pairs of objects (e.g., phone lines, persons, etc.); (b) Queries where aggregations are performed in groups of N objects, where N $>$ 2; (c) For (a) and (b), rollups to different dimension levels are performed.; (d) Graph OLAP-style aggregations performed over graph metrics. The idea of these experiments is to study if, when the queries can take advantage of the graph structure, graphoid-OLAP queries are more concisely expressed, and more efficiently executed. The impact of N in the relational and the graph representation is also studied. The queries are described next. For the sake of space, only some of the SQL and Neo4j queries are shown.

Query 1. *Average duration of the calls between groups of N phone lines.*

This query computes all the N-subsets of lines that participated in some call. That means, if a call involves 3 lines, say Ph_1, Ph_2 and Ph_3, and $N = 2$, the groups will be (Ph_1, Ph_2), (Ph_1, Ph_3), and (Ph_2, Ph_3). Figure 13 shows the recursive SQL query for the first representation alternative. The Cypher query for $N = 2$ is shown in Figs 14 and 15 shows the Cypher query for $N = 3$.

Query 2. *Average duration of the calls between groups of N users.*

Figure 16 shows the Cypher query for $N = 3$. Note that in all cases the queries actually perform a roll-up to the level User along the Phone dimension. The relational queries perform this roll-up through a join between the fact and dimension tables. In the case of Neo4j the roll-up is performed using by pattern matching. That is, the climbing (in the graphoid OLAP model) is done by the MATCH clause (the climbing path is explicit in this clause), while the aggregation is performed in the RETURN clause.

Query 3. *Average duration of the calls between groups of N operators.*

This analyses a roll-up to the level Operator, which has less instance members than the level User addressed in Query 2.

Table 3
Experimental results (running times in seconds)

Dataset	Calls	Calls	Calls	Calls-alt	Calls-alt	Calls-alt	Neo4j	Neo4j	Neo4j
	$N = 2$	$N = 3$	$N = 4$	$N = 2$	$N = 3$	$N = 4$	$N = 2$	$N = 3$	$N = 4$
D1-Q1	**4.9**	**7.6**	**9.5**	5.4	8.7	10.6	7.3	11.2	12.5
D1-Q2	4.6	**11.7**	**12.9**	**4.4**	12.3	14.5	7	**11.7**	14.8
D1-Q3	6.6	**7.3**	**11.5**	12.8	12.6	14.7	**3.7**	10.8	15.5
D1-Q4	∞	N/A	N/A	∞	N/A	N/A	**185**	N/A	N/A
D1-Q5	∞	N/A	N/A	∞	N/A	N/A	**21**	N/A	N/A
D1-Q6	∞	N/A	N/A	∞	N/A	N/A	**6**	N/A	N/A
D1-Q7	∞	N/A	N/A	∞	N/A	N/A	**34**	N/A	N/A
D2-Q1	**9.3**	**14.1**	15.1	10.4	16.2	17.7	15.6	17.5	21.6
D2-Q2	**12.9**	**19**	20.7	14.5	24	26.8	20.2	21.6	24.8
D2-Q3	12.5	19.4	22.2	14.3	**14.6**	22.8	**9.3**	18.7	28.4
D2-Q4	∞	N/A	N/A	∞	N/A	N/A	∞	N/A	N/A
D2-Q5	∞	N/A	N/A	∞	N/A	N/A	**677**	N/A	N/A
D2-Q6	∞	N/A	N/A	∞	N/A	N/A	**123**	N/A	N/A
D2-Q7	∞	N/A	N/A	∞	N/A	N/A	**924**	N/A	N/A

```
MATCH (m:Phone),(n:Phone)
WITH m,n WHERE m<>n
MATCH p= shortestPath((m)-[:receiver|:creator *]-(n))
RETURN p, length(p)
```

Fig. 17. Query 3 – Cypher expression.

Query 4. *For each pair of Phones in the Calls graph, compute the shortest path between them.*

This query aims at analysing the connections between phone line users, and has many real-world applications (for example, to investigate calls made between two persons who use a third one as an intermediary). From a technical point of view, this is an aggregation over the whole graph, using as a metric the shortest path between every pair of nodes. Figure 17 shows the corresponding Cypher query.

Finally, the following queries combine the computation of graph metrics together with roll-up and dice operations.

Query 5. *Compute the shortest path between pairs* (p_1, p_2) *of phone lines, such that* p_1 *corresponds to operator "Claro" and* p_2 *corresponds to operator "Movistar".*

Query 6. *Compute the shortest path between pairs* (p_1, p_2) *of phone lines, such that* p_1 *corresponds to a user from the city of Buenos Aires and* p_2 *corresponds to a user from the city of Salta.*

Query 7. *Compute the shortest path between pairs* (p_1, p_2) *of phone lines, such that* p_1 *corresponds to a user from the city of Buenos Aires.*

6.5. Results

Table 3 shows the results of the experiments. The tests were ran on machine with a i7-6700 processor and 12 GB of RAM, and 250 GB disk (actually, a virtual node in a cluster). The execution times are depicted, and are the averages of five runs of each experiment, expressed in seconds. The winning alternatives are marked in *boldface*, for clarity.

6.6. Discussion of results

In Table 3 it can be seen that running traditional OLAP queries, like Query 1, Query 2 and Query 3, takes approximately the same time in the relational and graphoid models, with a slight advantage for the former. Further, it can be seen that for Queries 2 and 3, which include a roll-up, results are very similar, and even Neo4j wins here in some cases. In Query 1, which is an aggregation over the fact graph, the relational alternatives work better.[13] However, for typical Graph OLAP queries (Queries 4 through 7), which aggregate graph metrics, *the graph model shows a dramatical advantage over the relational alternative*. For Neo4j, Query 4 does not finish within a reasonable time for the largest of the two datasets (D2) but performance is acceptable for D1. On the other hand, the relational alternatives do not terminate successfully neither for D1 nor for D2. It is important to make it clear that with an ad-hoc relational design, specifically for graph representation, it is possible that the performance of the relational alternative for shortest path aggregations could be improved, although it will hardly be close to the graph alternative, given the results presented here. However, the intention of this paper is to present a flexible model that can perform efficiently on a variety of situations. In this sense, the tests presented here suggest that the graphoid data model can be competitive with the relational model for classic OLAP queries, but is much better for typical Graph OLAP ones.

7. Conclusion and open problems

This paper presented a data model for graph analysis based on node- and edge-labelled directed multi-hypergraphs, called graphoids. A collection of OLAP operations, analogous to the ones that apply to data cubes, was formally defined over graphoids. It was also formally proved that the classic data cube model is a particular case of the graphoid data model. As far as the authors are aware of, this is the first proposal that formally addresses the problem of defining OLAP operations over hypergraphs. Supported by this proof, it was shown that the graphoid model can be competitive with the relational implementation of OLAP, but clearly much better when graph operations are used to aggregate graphs. This feature allows devising a general OLAP framework that may cope with the flexible needs of modern data analysis, where data may arrive in different forms. It is worth to remark, once more, that the experiments presented do not pretend to be exhaustive, but a good general indication of the plausibility of the approach, and it is clear that the graph data model provides OLAP with a machinery of more powerful tools than the classic cube data model, which is already good news for the OLAP practitioners.

Building on the results in this paper, future work includes looking for further graph metrics that can be applied to the graphoid model, new case studies, and the study of query optimization strategies. Moreover, the approach can also benefit from tools supporting parallel computation with columnar databases as backends. This can further improve the relational OLAP computation, while keeping the properties of the graphoid model for Graph OLAP queries.

Acknowledgments

Alejandro Vaisman was supported by a travel grant from Hasselt University (Korte verblijven-inkomende mobiliteit, BOF16KV09). He was also partially supported by PICT-2014 Project 0787 and PICT-2017 Project 1054. The authors also thank T. Colloca, S. Ocamica, J. Perez Bodean, and N. Castaño, for their collaboration in the data preparation for the experiments.

[13] It is worth noting that Neo4j (and graph databases in general) is a novel database, whose query optimization strategy is still very basic. On the contrary, relational databases are mature technologies, and query optimization is very efficient indeed. Further, for the experiments presented here, the PostgreSQL databases have been tuned to perform in the best possible way. In this sense, Neo4j's performance for typical OLAP queries is, in some sense, penalized.

References

[1] R. Angles, A Comparison of Current Graph Database Models, in: *Proceedings of ICDE Workshops*, Arlington, VA, USA, 2012, pp. 171–177.

[2] R. Angles, M. Arenas, P. Barceló, A. Hogan, J.L. Reutter and D. Vrgoc, Foundations of modern query languages for graph databases, *ACM Comput. Surv.* **50**(5) (2017), 68:1–68:40.

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

[4] S.-M.-R. Beheshti, B. Benatallah, H.R.M. Nezhad and M. Allahbakhsh, A framework and a language for on-line analytical processing on graphs, in: *WISE 2012*, Springer, 2012, pp. 213–227.

[5] 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) (2009), 41–63.

[6] J. Cohen, B. Dolan, M. Dunlap, J. Hellerstein and C. Welton, MAD Skills: New analysis practices for big data, *Proceedings of the VLDB Endowment* **2**(2) (2009), 1481–1492.

[7] A. Cuzzocrea, L. Bellatreche and I.-Y. Song, Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions, in: *Proceedings of DOLAP*, New York, NY, USA, ACM, 2013, pp. 67–70.

[8] B. Denis, A. Ghrab and S. Skhiri, A distributed approach for graph-oriented multidimensional analysis, in: *IEEE International Conference on Big Data*, 2013, pp. 9–16.

[9] L. Etcheverry and A. Vaisman, QB4OLAP: A vocabulary for OLAP cubes on the semantic web, in: *Proceedings of COLD*, Boston, USA, 2012. CEUR-WS.org.

[10] A. Ghrab, O. Romero, S. Skhiri, A.A. Vaisman and E. Zimányi, GRAD: Modeling and Querying Data Warehouses, in: *Proceedings of ADBIS*, Poitiers, France, 2015, pp. 92–105.

[11] L.I. Gómez, B. Kuijpers and A.A. Vaisman, Performing OLAP over graph data: Query language, implementation, and a case study, in: *Proceedings of BIRTE*, Munich, Germany, August 28, 2017, pp. 6:1–6:8.

[12] L.I. Gómez, B. Kuijpers and A.A. Vaisman, Analytical queries on semantic trajectories using graph databases, *TGIS Trans. Geog. Inf. Syst.* **23**(5) (2019).

[13] O. Hartig, Reconciliation of RDF* and property graphs, CoRR, abs/1409.3288, 2014.

[14] H. He and A. Singh, Query language and access methods for graph databases, in: *Managing and Mining Graph Data, Vol. 40 of Advances in Database Systems*, Springer US, 2010, pp. 125–160.

[15] R. Kimball, The Data Warehouse Toolkit, J. Wiley and Sons, 1996.

[16] M.B. Kraiem, J. Feki, K. Khrouf, F. Ravat and O. Teste, Modeling and OLAPing social media: the case of twitter, *Social Netw. Analys. Mining* **5**(1) (2015), 47:1–47:15.

[17] B. Kuijpers and A.A. Vaisman, An algebra for OLAP, *Intelligent Data Analysis* **21**(5) (2017).

[18] Q. Qu, F. Zhu, X. Yan, J. Han, P.S. Yu and H. Li, Efficient topological OLAP on information networks, in: *Proceedings of DASFAA – Vol. Part I*, Springer, 2011, pp. 389–403.

[19] N.U. Rehman, A. Weiler and M.H. Scholl, OLAPing social media: the case of twitter, in: *Advances in Social Networks Analysis and Mining 2013, ASONAM '13*, Niagara, ON, Canada, 2013, pp. 1139–1146.

[20] I. Robinson, J. Webber and E. Eifrém, Graph Databases, O'Reilly Media, 2013.

[21] M. Rudolf, H. Voigt, C. Bornhövd and W. Lehner, Synopsys: Foundations for multidimensional graph analytics, in: *Proceedings of BIRTE*, Hangzhou, China, Revised Selected Papers, September 1, 2014, pp. 159–166.

[22] B. Tang, S. Han, M.L. Yiu, R. Ding and D. Zhang, Extracting top-k insights from multi-dimensional data, in: *Proceedings of ACM SIGMOD*, Chicago, IL, USA, May 14–19, 2017, pp. 1509–1524.

[23] Y. Tian, R.A. Hankins and J.M. Patel, Efficient aggregation for graph summarization, in: *Proceedings ACM SIGMOD*, ACM, 2008, pp. 567–580.

[24] A. Vaisman, F. Besteiro and M. Valverde, Modelling and querying star and snowflake warehouses using graph databases, in: *Proceedings of ADBIS Conference 2019*, Bled, Slovenia, Sept. 8–11, 2019.

[25] A.A. Vaisman, Publishing OLAP cubes on the semantic web, in: *EBIS'15, Lecture Notes in Business Information Processing*, Springer, 2015, pp. 32–61.

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

[27] P. Wang, B. Wu and B. Wang, TSMH graph cube: A novel framework for large scale multi-dimensional network analysis, In: *IEEE DSAA*, Paris, France, October 19–21, 2015, pp. 1–10.

[28] 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: *Proceeding of IEEE ICDE*, 2014, pp. 496–507.

[29] X. Wu, B. Wu and B. Wang, P&D graph cube: Model and parallel materialization for multidimensional heterogeneous network, in: *Proceeding of CyberC 2017*, Nanjing, China, October 12–14, 2017, pp. 95–104.

[30] M. Yin, B. Wu and Z. Zeng, HMGraph OLAP: a novel framework for multi-dimensional heterogeneous network analysis, in: *Proceedings of DOLAP*, ACM, 2012, pp. 137–144.

[31] P. Zhao, X. Li, D. Xin and J. Han, Graph Cube: on warehousing and OLAP multidimensional networks, in: *Proceedings of ACM SIGMOD*, ACM, 2011, pp. 853–864.