OLAPing Graph Data

Leticia Gómez[1], Bart Kuijpers[2], Alejandro Vaisman[3]

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. In spite of this, 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. Graphoids can express, in a natural way, situations than imply relations between a variable number of dimensions, which is not easily done in the classical relational OLAP model. 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 remains competitive.

Keywords: OLAP, Data Warehousing, Graph Database, Big Data, Graph Aggregation

[1]Instituto Tecnológico de Buenos Aires, Buenos Aires, Argentina; email: lgomez@itba.edu.ar

[2]Hasselt University, Belgium; email: bart.kuijpers@uhasselt.edu

[3]Instituto Tecnológico de Buenos Aires, Buenos Aires, Argentina; email: avaisman@itba.edu.ar (Corresponding author).

1 Introduction

Online Analytical Processing(OLAP) [14, 22] comprises a set of tools and algorithms that allow querying multidimensional (MD) databases. In these databases, data are modelled as *data cubes*, where each cell contains one or more *measures* of interest, that quantify *facts*. Measure values can be aggregated along *dimensions*, organized as sets of hierarchies. Traditional OLAP queries are used to manipulate the data cube: 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, are among the most popular OLAP operations on cube data. 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 the use of OLAP for social network analysis has been proposed [19, 15], in a "Big Data" scenario, the classic OLAP process can still be useful, but other requirements appear, as claimed in the classic paper by Cohen et al. [6], where the so-called MAD skills (standing from Magnetic, Agile and Deep) required for data analytics are described. This calls for a new vision of the data acquisition problem in DW, normally denoted ELT (Extract, Load, Transform), where data are first loaded, and then modelled, if required. In this scenario, more complex analysis tools are required, that go beyond the data cube model, for example, graph analytics. Graphs, and, particularly, property graphs [20, 12], are becoming increasingly popular to model different kinds of networks (for instance, social networks, sensor networks, and the kind) to perform data analysis, and 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), Sparksee[3], and GraphFrames[4]. In addition to traditional graph analytics, it is also interesting for the data scientist to have the possibility of performing OLAP on graphs. Thus, novel models and techniques are required for enabling MD analysis of Big Data [7].

From the discussion above, the following challenges arise: 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 these challenges, not

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

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

[3]`http://www.sparsity-technologies.com/#sparksee`

[4]`https://graphframes.github.io/`

many proposals so far have been presented in this sense, e.g. [5, 10, 24, 26]. Most of these address homogeneous graphs (that is, graphs where all nodes are of the same type), and OLAP analysis focus mainly on the graph topology as the measure of interest [5, 24, 26]. In such cases, all the attributes of the graph elements are considered as the dimensions, and are used for aggregating the graph and performing its multi-perspective analysis. However, real-world graphs are complex and often heterogeneous, that is, graphs where nodes and edges could be of different types to represent different real-world entities, and the different relationships between them. In [10], a technique for building OLAP cubes on (possibly heterogeneous) graphs is proposed. These works will be discussed in more detail in Section 2, and compared against the approach introduced in the present paper.

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). Unlike relational OLAP, the hypergraph model makes it natural the 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 a typical "Star" or "Snowflake" representation schema [14] 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, the relational representation (i.e., ROLAP) works fine, but when this is not the case, modelling and querying issues appear, which call for a more natural representation, closer to the original data format. And here is where the hypergraph model comes to the rescue, as this paper shows.[5] As an additional feature, the graph approach can adapt to many different analysis settings. That is, graphoids can be

[5]An extended abstract of this paper has been presented in [11]. The present works substantially extends such abstract, with a completely different running example and case study, the proof of the main theorem, and the implementation of the proposal together with experimental results that shows its plausibility. All in all, it can be seen that the published abstract has less than a 20% overlap with the work presented here.

used on an ETL scenario, where data are carefully modelled and cleaned, and directly exported from a DW to a graph. Or they could be applied on an ELT scenario, where, for instance, data are loaded into so-called "data lakes"[6], with minimum preparation. 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.

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: in Section 2 we discuss related work. Section 3 presents the graphoid data model. Section 4 presents the OLAP operations on graphoids, while the main contribution of Section 5 consists in showing 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 and discusses open problems.

[6]`https://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/`

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 (each attribute value is an element in the domain of a dimension level), and thus nodes and edges can be aggregated, provided 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.

Graph Database Modelling Graph database modelling and querying are the foundations for the graphoid OLAP model. 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 [13]) 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[7], oriented to the Semantic Web. This is the case of AllegroGraph.[8] Given this characteristic, this is the only graph database which supports reasoning.

(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 of RDF makes them not as efficient as the other models, which are aimed at reaching a local scope. An

[7]`https://www.w3.org/RDF/`

[8]`http://franz.com/agraph/allegrograph/`

important feature of RDF-base graph models, however, is that they follow a standard, which is not yet the case for the other graph databases. Hartig [12] 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[9], the standard query language for the Semantic Web.

In [4], the authors introduced a framework for OLAP on RDF data. They proposed GOLAP, a graph model for OLAP on graphs, and FSPARQL an extension to SPARQL for OLAP querying of RDF data. GOLAP has the particularity of supporting 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, 23], where the QB4OLAP vocabulary is the basis for defining the OLAP operations on RDF graphs.

The present paper works with models based on property graphs.

Graph Summarization and OLAP Graph summarization is a critical operation for multi-level analysis of graph data. Tian et al. [21] 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 presents an algorithm to evaluate the SNAP operation, and shows that the k-SNAP computation is NP-complete, proposing 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 discussed the structural

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

aggregation of the graph following the OLAP paradigm, presenting techniques based on the properties of the graph measures for optimizing measure computation through the different aggregation levels.

GraphCube [26] is a framework for OLAP cubes computation and analysis through the different levels of aggregation of a graph. It targets *single, homogeneous, node-attributed graphs.* The framework introduces so-called cuboid and crossboid queries, for building and analyzing the different graph cubes. Along similar lines, Pagrol [24] is a Map-Reduce 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 behind these two proposals is basically the same: to efficiently compute all possible aggregations of an homogeneous graph. Also based on the notion of so-called graph cuboids (that is, graphs defined at different levels of aggregation), Distributed Graph Cube [8] is a distributed framework for graph cube computation and aggregation, implemented using Spark[10] and Hadoop[11]. Again, this proposal only supports *homogeneous* graphs. To handle heterogeneous graphs, and also starting from the topological and informational dimensions introduced in [5], Yin et al. [25] propose 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.

More oriented to 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.

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. At the same time, this increases the difficulty of the problem addressed. Now, instead of binary relationships between nodes, there are n-ary, probably duplicated relationships. In other words, the model handles multi-hypergraphs, which are the graphs typically found in real-world "Big data" scenarios. 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, regardless the kind of underlying data (in this case, graphs), since these operations could be given, conceptually, in

[10]http://spark.apache.org/
[11]http://hadoop.apache.org/

terms of cubes and dimensions rather than nodes and edges.

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.

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 [16, 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 set), 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 to 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 $\sigma(Phone)$ and $\sigma(Time)$, 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

$$\text{Phone} \rightarrow \text{Customer} \rightarrow \text{City} \rightarrow \text{Country} \rightarrow \mathit{All},$$

and

$$\text{Phone} \rightarrow \text{Operator} \rightarrow \mathit{All}.$$

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 $\rightarrow$ Month $\rightarrow$ Year $\rightarrow$ *All*. $\square$

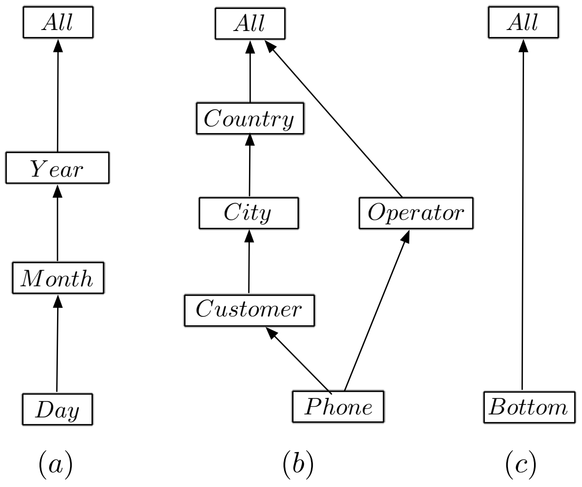


Figure 1: Dimension schemas for the dimensions Time (a), Phone (b), and Id (identifier) (c).

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

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

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

where the union is taken over all levels in $\sigma(D)$. The edge set of this directed acyclic graph is defined as follows. Let ℓ and ℓ' be two levels of $\sigma(D)$, and let $a \in dom(D.\ell)$ and $a' \in dom(D.\ell')$. Then, only if there is a directed edge from ℓ to ℓ' in $\sigma(D)$, there can be a directed edge in $I(\sigma(D))$ from a to a'.

If H is a hierarchy in $\sigma(D)$, then the *hierarchy instance* (relative to the dimension instance $I(\sigma(D)))$ is the subgraph of $I(\sigma(D))$ with nodes from $dom(D.\ell)$, for ℓ appearing in H. This subgraph is denoted $I_H(\sigma(D))$. $\square$

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 Continuing with Example 1, consider dimension Phone, whose schema $\sigma(Phone)$ is given in Figure 1 (b).

Associated with this schema, there is an instance where $dom(Phone) = dom(Phone.Bottom) = dom(Phone.Phone) = \{Ph_1, Ph_2, Ph_3, Ph_4, Ph_5\}$. For level Operator, $dom(Phone.Operator) = \{ATT,\ Movistar,\ Vodafone\}$. This dimension instance $I(\sigma(Phone))$ is depicted in Figure 2, which shows, e.g., that phone lines Ph_2 and Ph_4 correspond to the operator $Vodafone$. □

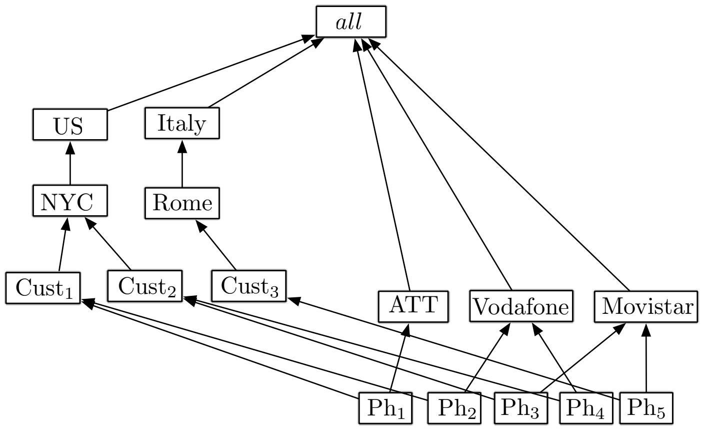


Figure 2: An example of a dimension instance $I(\sigma(Phone))$ for the dimension $Phone$.

In a dimension graph with multiple hierarchies, elements in some levels may be reachable from elements in the *Bottom* level, in multiple ways. In what follows, "sound" dimension graphs are assumed. These graphs guarantee that rolling-up in different ways (that is, through different paths) gives the same results [16, 17], typical in so-called balanced (or homogeneous) dimensions [22].

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 is designed to contain 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 $D_1, ..., D_d$ is assumed in the application domain, and their schemas $\sigma(D_1), ..., \sigma(D_d)$ are given. Furthermore, hierarchy instances $I(\sigma(D_1)), ..., I(\sigma(D_d))$ for all dimensions are given. Finally, assume that a special dimension $D_0 = \text{Id}$ is given, to represent unique identifiers (Figure 1(c)). Prior to defining graphoids at levels $\ell_1, ..., \ell_d$ in the respective dimensions, the notions of *attributes*, *node types* and *edge types* are needed.

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

Node types Assume a finite, non-empty set $\mathcal{N}$ of *node types.* Elements of $\mathcal{N}$ will be 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. Also assume the existence of the functions *ar* and *dim* 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. Secondly, $dim(\#\text{n})$ is an $ar(\#\text{n})$-tuple of attributes, which are dimensions (at Bottom-level), the first of which is the Identifier dimension. This means that $dim(\#\text{n})$ is an element of $\{\text{Id}\} \times \{D_1, ..., D_d\}^{ar(\#\text{n})-1}$. The tuple $dim(\#\text{n})$ expresses which attributes are associated with a node of type #n, without specifying their levels. Finally, assume that $dim(\#\text{n})$ contains no repetition, which is the usual case in practice. The identifier dimension is always used at its *Bottom* level.

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 such that they participated 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 that is associated with an edge of type #e. Secondly, $dim(\#\text{e})$ is an $ar(\#\text{e})$-tuple of attributes, which are dimensions (at Bottom-level). This means that $dim(\#\text{e})$ is an element of $\{D_0, D_1, ..., D_d\}^{ar(\#\text{e})}$. The tuple $dim(\#\text{n})$ expresses which attributes are associated with an edge of type #e, without specifying their levels. Finally, assume that $dim(\#\text{e})$ contains no repetition. The identifier dimension (at its *Bottom* level) may appear, but is not required. If the identifier dimension appears, it only appears once among the attributes that describe edges of a certain type.

It is now possible to define the notion of "graphoid", which will serve as the basic data structure for the graph-OLAP process.

Definition 3 (Graphoid) Let $D_0 = \text{Id}$ be the identifier dimension. Let dimensions $D_1, ..., D_d$ be given with their respective schemas and instances. Let $\ell_1, ..., \ell_d$ be levels for these respective dimensions. A $(D_1.\ell_1, ..., D_d.\ell_d)$-*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 $[\#\text{n}, a_1, ..., a_{ar(\#\text{n})}]$, where $\#\text{n} = \tau_N(n)$ and, if $dim(\#\text{n}) = (A_1, ..., A_{ar(\#\text{n})})$, then, for $i = 1, ..., ar(\#\text{n})$, $a_i \in dom(D_j.\ell_j)$, 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[12] of the set $\mathcal{P}(N) \times \mathcal{P}(N)$, which we call the set of *(multi hyper-)edges* of G;
- 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 $[\#\text{e}, b_1, ..., b_{ar(\#\text{n})}]$, where $\#\text{e} = \tau_E(e)$ and, if $dim(\#\text{e}) = (B_1, ..., B_{ar(\#\text{n})})$, then, for $i = 1, ..., ar(\#\text{e})$, $b_i \in dom(D_j.\ell_j)$, if B_i is the dimension D_j; λ_E is called the *edge labelling function.* $\square$

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.

[12]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 $A \subseteq B$.

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

Example 3 (Base graph) 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 will be shown, represented as a graph. The Phone dimension can play the roles of the calling line and the callee line (this is called a role-playing dimension in the OLAP literature [22]). The information of 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 #Phone has arity 2. Its first attribute is a node identifier and the second attribute is a dimensional one that represents the phone number, with domain $\{\text{Ph}_1, \text{Ph}_2, ...\}$. In the example of Figure 3,

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

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 #Call and $\mathcal{E} = \{\#\mathsf{Call}\}$. In Figure 3, a directed hype-edge 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 $S \to T$ from now on.[13] For example, the red and purple hyperedges $\{1\} \to \{2\}$ 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 sets. The orange hyperedge $\{3\} \to \{2, 5\}$ represents a group call, from Ph_3 to both Ph_2 and Ph_5. In the figure, we see six phone calls. So, E is the bag $\{\!\{\{1\} \to \{2\}, \{1\} \to \{2\}, \{4\} \to \{3\}, \{4\} \to \{5\}, \{3\} \to \{2, 5\}, \{5\} \to \{2, 3\}\}\!\}$. The edge labelling function λ_E associates two attributes, with edges

[13]The nodes of S are called the *source nodes* of e and the nodes of T are called the *target nodes* of e. If S has one element, then the hyperedge e is called *single source*. The source and target nodes of e are called *adjacent* to e, and the set of the adjacent nodes to e are denoted by $Adj(e)$. Thus, $Adj(e) = S \cup T$.

of type #Call, namely Date and Duration. Date is a dimensional attribute to which the dimensional hierarchy in Figure 1 is associated. Duration is a measure attribute (which has as an associated aggregation function, in this case, the summation).

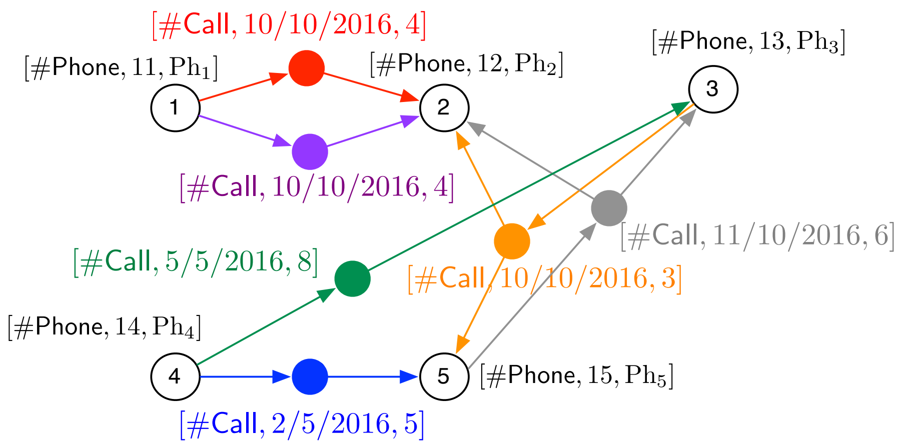


Figure 3: Basic phone call data as a base graph.

□

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 in the paper, it will be explained how these graphoids can be obtained from the base one.

Example 4 (Graphoid) 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 Figure 3, is shown in Figure 4. Here, in the Phone nodes, the phone numbers (e.g., Ph_3) have been replaced with their corresponding operator name, at the Phone.Operator level in the dimension Phone (for example, for Ph_3, the corresponding operator is Movistar).

Figure 5 shows an alternative (Time.Day, Phone.Operator)-graphoid for the data from Figure 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 Figure 4), which belong

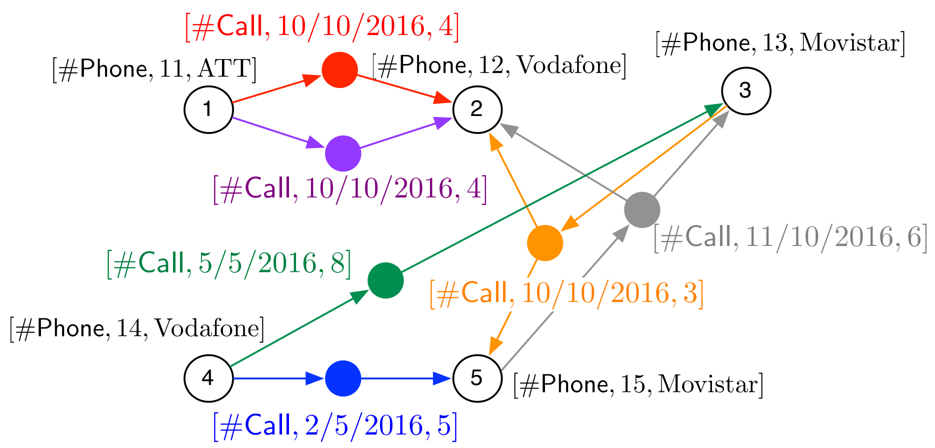


Figure 4: A (Time.Day, Phone.Operator)-graphoid, based on the data shown in Figure 3.

to the operator Vodafone. 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 Figure 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 is more than one (Time.Day, Phone.Operator)-graphoid that is "consistent" with the given base graph. This suggests that some kind of normalization is needed. This is studied in the next section.

□

Remark 3 The conceptual data modelling view underlying this proposal requires some additional explanation, which is given next.

Nodes are assumed to represent basic objects in the modelled application world. These objects are given by a number of descriptive attributes. Measure information, that is 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

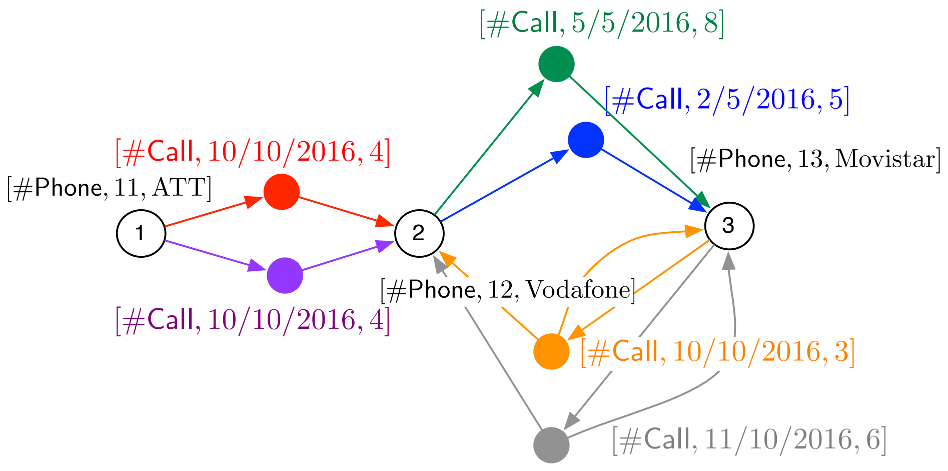


Figure 5: An alternative (Time.Day, Phone.Operator)-graphoid, based on the data shown in Figure 3.

phone number, for example, [#Phone, 11, 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 Figure 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 Edgify(#n, A).

This is consistent with both possibilities of data acquisition and modelling, namely ETL and ELT, commented in Section 1. In the latter, mostly used in a "Big Data" scenario, it is usual to capture data as they come, and then decide whether or not to model these data in a particular way. Edgification is useful in these scenarios. When it is possible to model data prior to exploitation, the choice would most likely be to represent measures as edges. However, when data are acquired as they come, it is most likely that measures like the ones above, come as attribute of some node or dimension. Then, the notion of edgification gives the possibility of changing this situation on-the-fly. □

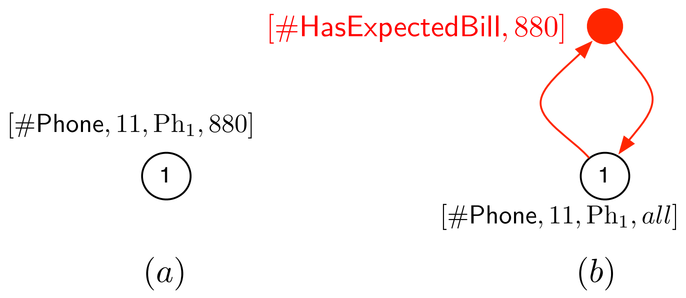


Figure 6: (a) A node with label $[\#\mathsf{Phone}, 11, \mathrm{Ph}_1, 880]$, where 880 expresses the expected bill. (b) An edgification of this node, where the expected billing information is moved to an edge that is labelled $\#\mathsf{HasExpectedBill}$.

3.3 Minimal graphoids

Example 4 showed that Definition 3 allows the existence of more than one $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid. In this section, the notion of *minimal* $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid is defined. This graphoid is obtained from a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid by collapsing nodes which (apart from the identifier) have identical labels (that is λ_N-values).

Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid. If the nodes $n_1, n_2 \in N$ have identical labels, apart from the identifier (which by definition are distinct), denoted $\lambda_N(n_1) =_{\mathrm{Id}} \lambda_N(n_2)$, then these nodes are identified. This means that only the one with the smallest identifier is preserved, while the others are deleted. So, if the λ_N-values of the nodes $n_1, n_2, ..., n_k$ pairwise satisfy the $=_{\mathrm{Id}}$-relationship, and n_1 has the smallest identifier among them, then the nodes $n_2, ..., n_k$ are replaced by n_1 and then deleted. The expression $rep_N(n_i) = n_1$, for $i = 1, 2, ..., k$, indicates that n_1 will represent the nodes $n_1, n_2, ..., n_k$ in the minimal graph. All edges leaving from or arriving at the nodes $n_2, ..., n_k$ will be 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 $D_0, D_1, ..., D_d$ and $\ell_1, ..., \ell_d$ be the same as in Definition 3. Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid. The *minimal graphoid of* G is the $(D_1.\ell_1, ..., D_d.\ell_d)$-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.

$\square$

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 from 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. $\square$

Example 5 (Minimal graphoid) Figures 4 and 5 in Example 4 depict two (Time.Day, Phone.Operator)-graphoids that correspond to the graph of Figure 3. The graphoid of Figure 5 is the minimal graphoid of Figure 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 as it is. Between the 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 the 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 the nodes 2, 3 and 5 correspond to two hyperedges between the nodes 2 and 3 in the minimal graph. $\square$

Proposition 1 below, immediately follows from the definition of minimal graphoid.

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

$\square$

Definition 6 For any $(D_1.\ell_1, ..., D_d.\ell_d)$-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 $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid of a $(D_1.\ell_1, ..., D_d.\ell_d)$-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 $|E|$. However, this can be improved in most cases, for instance, with an early pruning of the nodes that will not be contracted. Addressing these issues 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. In Section 5 it is shown that these operations can simulate the typical OLAP operations on cubes, when data cubes are represented as graphs.

4.1 Climb

First, the Climb-operation, both on nodes and on edges is introduced. Intuitively, this operation allows to define graphs at different levels of granularity, based on the background dimensions.

Definition 7 Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid. 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 $\ell_k \to \ell'_k$. Also, let $\rho_{\ell_k \to \ell'_k}$ be the corresponding rollup function (at the instance level). 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-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 $\mathsf{Climb}(G, \#\mathsf{n}, D_k.(\ell_k \to \ell'_k))$, gives as a result the replacement of any attribute value a from $dom(D_k.\ell_k)$ by the new value $\rho_{\ell_k \to \ell'_k}(a)$ from $dom(D_k.\ell'_k)$, in all nodes 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 $\mathsf{Climb}(G, \#\mathsf{e}, D_k.(\ell_k \to \ell'_k))$, gives as a result the replacement of any attribute value a from $dom(D_k.\ell_k)$ by the new value $\rho_{\ell_k \to \ell'_k}(a)$ from $dom(D_k.\ell'_k)$, in all edges of G of type $\#\mathsf{e}$, leaving G unaltered otherwise. □

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

Remark 6 Often, in applications, a dimension D_k may appear in multiple node types and edge types, and the user may want to apply the Climb-operation on any number of them. In this case, the shorthand $\mathsf{Climb}(G, \{\#\mathsf{n}_1, ..., \#\mathsf{n}_r, \#\mathsf{e}_1, ..., \#\mathsf{e}_s\}, D_k.(\ell_k \to \ell'_k))$ is used, to indicate a climbing, in the dimension D_k, from level ℓ_k to level ℓ'_k in all the nodes of types $\#\mathsf{n}_1, ..., \#\mathsf{n}_r$, and in all the edges of types $\#\mathsf{e}_1, ..., \#\mathsf{e}_s$ in G. It is remarked that this can be performed in any order. The expression $\mathsf{Climb}(G, *, D_k.(\ell_k \to \ell'_k))$ 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 8 Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid. 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 Figure 4 the operation $\mathsf{Group}(G, \#\mathsf{Phone}, \mathrm{Phone}.(\mathrm{Phone} \to \mathrm{Operator}))$, results in the graphoid, depicted in Figure 5. □

4.3 Aggregate

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

Definition 9 Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a minimal $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid. Let D_k be a dimension that appears in the hyperedges of G of type $\#\mathsf{e}$, and 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 $\mathsf{Aggr}(G, \#\mathsf{e}, D_k, F_k)$, gives as result a graphoid G' over the same N, τ_N and λ_N as G, with the following modified hyperedge bag E'. If the hyperedges $e_1, e_2, ..., e_r$ are all of type $\#\mathsf{e}$ and all of type $S \to T$ (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 $e_1, e_2, ..., e_r$ 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 $e_1, e_2, ..., e_r$. $\square$

Example 8 Applying the operation $\mathsf{Aggr}(G, \#\mathsf{Call}, \text{Duration}, \textsc{Sum})$ to the graphoid G, depicted in Figure 5, results in the same graphoid, in which 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. $\square$

Remark 7 To aggregate multiple dimensions $M_1, ..., M_k$, using the aggregate functions $F_1, ..., F_k$ simultaneously, the notation would be: $\mathsf{Aggr}(G, \#\mathsf{e}, \{M_1, ..., M_k\}, \{F_1, ..., F_k\})$.

Also, for simplicity, only the typical SQL aggregate functions Sum, Max,Min and Count are considered. $\square$

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. This does not mean that the implementation of a Roll-up operation requires the sequence of operations described in Definition 4.4 below. $\square$

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 10 Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid. Let D_c be a dimension that appears in some nodes and/or hyperedges of G. This dimension plays the role of a climbing dimension. Let $M_1, ..., M_k$ 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 $F_1, ..., F_k$ are associated with them. Let $\#\mathsf{n}_1, ..., \#\mathsf{n}_r$ be node types appearing in G, and let $\#\mathsf{e}_1, ..., \#\mathsf{e}_s$ be hyperedge types appearing in G. The *roll-up of G over the dimensions $M_1, ..., M_k$ (using the functions $F_1, ..., F_k$) 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 $\#\mathsf{n}_1, ..., \#\mathsf{n}_r$ and edges of types $\#\mathsf{e}_1, ..., \#\mathsf{e}_s$, denoted

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

is defined as

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

□

Example 9 Applying to the graphoid depicted in Figure 5 the operation $\mathsf{Roll\text{-}Up}(G, \{\#\mathsf{Call}\}, \mathrm{Time}.(\mathrm{Day} \to \mathrm{Year}); \#\mathsf{Call}, \mathrm{Duration}, \textsc{Sum})$, results in the graphoid of Figure 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. □

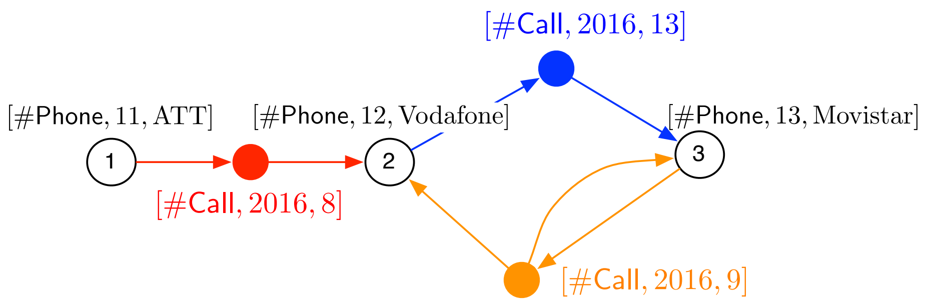


Figure 7: The result of the operation $\mathsf{Roll\text{-}Up}(G, \{\#\mathsf{Phone}\}, \mathrm{Time}.(\mathrm{Day} \to \mathrm{Year}); \#\mathsf{Call}, \mathrm{Duration}, \textsc{Sum})$ applied to the graphoid of Figure 5.

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: $\mathsf{Roll\text{-}Up}(G, *, D_c.(\ell_c \to \ell'_c); \#\mathsf{e}, M_1, ..., M_k, F_1, ..., F_k)$. Furthermore, to aggregate over all edge types, the notation becomes $\mathsf{Roll\text{-}Up}(G, *, D_c.(\ell_c \to \ell'_c); *, M_1, ..., M_k, F_1, ..., F_k)$. □

4.5 Drill-Down

The Drill-Down-operation does the opposite of the Roll-Up one,[14] 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, analogously to the Roll-Up-operation, the *drill-down of G over the dimensions $M_1, ..., M_k$ (using the functions $F_1, ..., F_k$) 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 $\#\mathsf{n}_1, ..., \#\mathsf{n}_r$ and edges of types $\#\mathsf{e}_1, ..., \#\mathsf{e}_s$*, denoted

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

is defined as

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

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

4.6 Dice

In this section, the Dice-operation over a graphoid is defined. This operation 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 $c \in dom(D.\ell)$. The expression φ can be written in disjunctive normal form as

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

where all φ_{kl} are atomic conditions.

[14]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.

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 in the usual way, it is sufficient to define $e \models \varphi_{kl}$ for the atomic formulas that appear in φ. Thus, the formula 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 that is adjacent to e (that is, $n \in Adj(e)$) is defined in a similar way. Finally, $e \models \varphi_{kl}$ if φ_{kl} is not false in e and not false in all $n \in Adj(e)$.

Definition 11 Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid. Let φ be a Boolean combination of equality and inequality constraints that involve, on the one hand dimension levels $\ell'_1, ..., \ell'_d$ (equal or higher than $\ell_1, ..., \ell_d$ in the dimension schemas $\sigma(D_1), ..., \sigma(D_d)$, respectively), and on the other hand, constants from $dom(D_1.\ell'_1), ..., dom(D_d.\ell'_d)$. 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 $\mathsf{s\text{-}Dice}(G, \varphi)$. $\square$

Example 10 Applying the operation $\mathsf{Dice}(G, \text{Phone.Operator} \neq \text{ATT})$ to the graphoid depicted in Figure 5, results in the graphoid of Figure 8(a). 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 Figure 8(a), results in the graphoid shown in Figure 8(b). $\square$

4.7 Slice

The description of the graph-OLAP operations concludes defining the Slice-operation over dimensions and measures stored in edges. Intuitively, this operation eliminates the references to a dimension in the graphoid. The analogy with classic OLAP slice operation will be discussed in Section 5.

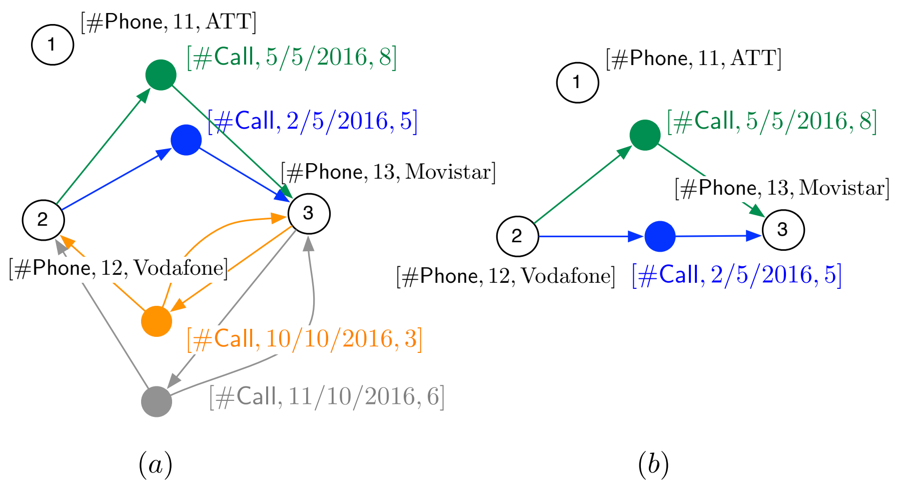


Figure 8: (a) The result of the operation $\mathsf{Dice}(G, \text{Phone.Operator} \neq \text{ATT})$ to the graphoid depicted in Figure 5; (b) The result of applying $\mathsf{Dice}(G, \text{Time.Month} = 5/2016)$ to (a).

Definition 12 Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid. Let D_s be a dimension that appears in some nodes and/or hyperedges of G. Let $M_1, ..., M_k$ be dimensions that appear in the hyperedges of G. These dimensions play the role of measure dimensions. It is assumed that aggregate functions $F_1, ..., F_k$ are associated with them. The *slice of the dimension D_s from G over the dimensions $M_1, ..., M_k$ (using the functions $F_1, ..., F_k$)*, denoted

$$\mathsf{Slice}(G, D_s; M_1, ..., M_k, F_1, ..., F_k),$$

is defined as the roll-up operation up to the level $D_s.All$ over the dimensions $M_1, ..., M_k$ (using the functions $F_1, ..., F_k$). Formally, this slice operation is defined as

$$\mathsf{Roll\text{-}Up}(G, *, D_s.(\ell_s \to All); *, M_1, ..., M_k, F_1, ..., F_k).$$

$\square$

Example 11 Applying the operation $\mathsf{Slice}(G, \text{Time}; \text{Duration}, \textsc{Sum})$ to the graphoid depicted in Figure 5, results in the graphoid of Figure 9. $\square$

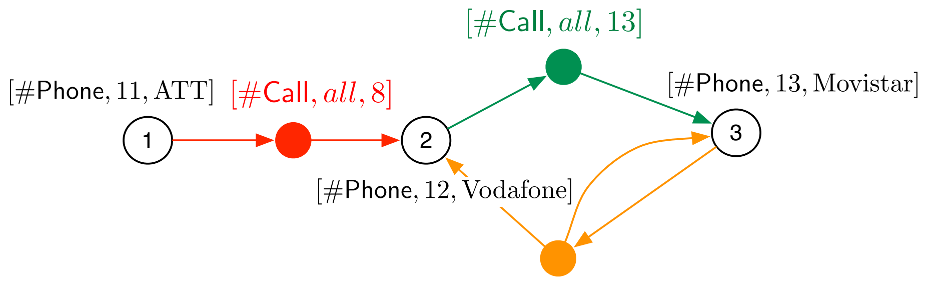


Figure 9: The result of the operation $\mathsf{Slice}(G, \text{Time}, \text{Duration}, \textsc{Sum})$ on the graphoid of Figure 5.

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 13 Let $G = (N, \tau_N, \lambda_N, E, \tau_E, \lambda_E)$ be a $(D_1.\ell_1, ..., D_d.\ell_d)$-graphoid. The *node-delete over G, given a node type* $\#\mathsf{n}$, denoted

$$\mathsf{n\text{-}Delete}(G, \#\mathsf{n}),$$

produces a subgraphoid of G, whose nodes of type $\#\mathsf{n}$ are deleted, and in which 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 $\#\mathsf{n}$. The edges remain of the same type and they keep the same label. □

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

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 Preliminary Discussion on Modelling

Consider a typical example of an OLAP cube with dimensions $(D_1, D_2, D_3) = (Product,\ Location,\ Time)$. Such a cube, illustrated in Figure 10, represents sales amounts of products at certain stores locations (cities) on certain dates (at the lowest level of granularity). There are several ways in which such a cube can be represented in the graphoid model. For instance, Figure 11 shows two alternative 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.

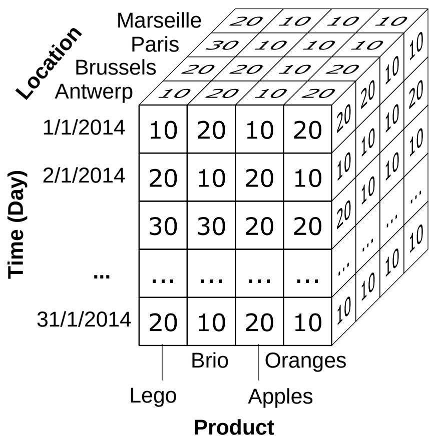


Figure 10: An example of a Sales data cube with one measure: $\mu_1 = sales$.

Figure 11(a) shows nodes 1, 2 and 3, of types #Product, #Location and #Time, respectively. All of these nodes 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, which is 10. So, in this approach, each cell of a data cube is modelled by a "star"-shaped hyperedge.

An alternative representation is shown in Figure 11(b), which is the most compact representation. In this approach there is only one node, of type #Cube in the graphoid, which represents the data cube. This node is labelled [#Cube, 11] in this example. It has no attribute values (apart from an identifier value). Cell-coordinates and cell-content are stored in hyperedges that form loops in the only node. The fact (*Lego*, *Antwerp*, *1/1/2014*; 10) is modelled by a unique hyperedge with label [#InCube, Lego, Antwerp, 1/1/2014, 10]. So, 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.

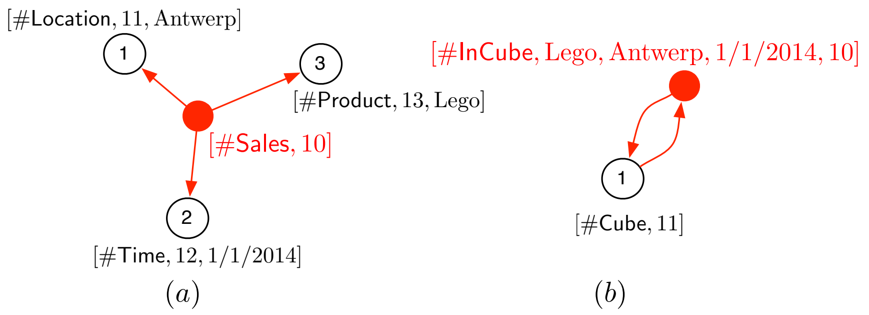


Figure 11: (a) The star-representation of the fact $(Lego, Antwerp, 1/1/2014; 10)$. (b) The petal-representation of the fact $(Lego, Antwerp, 1/1/2014; 10)$.

The next section will show that the graphoid OLAP-operations (as 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 (Figure 11(a)). Also, the semantics of the classical OLAP operations, as defined in [16, 17], is used.

5.2 Graph- and Classic- OLAP Operations Equivalence

A classical data cube is based on dimensions $D_1, ..., D_d$ and on measures $\mu_1, ..., \mu_m$. 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 the dimensions $D_1, ..., D_d$ with measures* $\mu_1, ..., \mu_m$, can then be seen as a (possibly partial) function $\mu : dom(D_1) \times \cdots \times dom(D_d) \to dom(\mu_1) \times \cdots \times dom(\mu_m)$. This function maps each "cell" of the cube to m values for the measures. We denote a cell of the cube with coordinates $(a_1, ..., a_d) \in dom(D_1) \times \cdots \times dom(D_d)$, that contains values $(c_1, ..., c_m) \in dom(\mu_1) \times \cdots \times dom(\mu_m)$, simply by $(a_1, ..., a_d; c_1, ..., c_m)$.

The "star-representation" of a data cube in the graphoid model can be formally defined now.

Definition 14 Let C be a data cube over the dimensions $D_1, ..., D_d$ with measures $\mu_1, ..., \mu_m$. The *star-graphoid* of C, denoted $\mathsf{Star}(C)$, is the following graphoid.

- For $i = 1, ..., d$, for each $a_i \in dom(D_i)$, 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 $(a_1, ..., a_d; c_1, ..., c_m) \in dom(D_1) \times \cdots \times dom(D_d) \to dom(\mu_1) \times \cdots \times dom(\mu_m)$, there are m hyperedges: for each $j = 1, ..., m$, 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 $i = 1, ..., d$, which is labelled $[\#\mu_j, c_j]$. $\square$

Now, the main theorem of this section is stated.

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

Proof 1 *Let C be a data cube and let $\mathsf{Star}(C)$ be its corresponding star-graphoid. The proof of the theorem is based on showing that each of the classical OLAP operations* Roll-Up, Drill-Down, Slice *and* Dice, *over C, can be equivalently applied on* $\mathsf{Star}(C)$.

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 $\mu_1, ..., \mu_m$, using aggregate functions $F_1, ..., F_m$. Assume, without loss of generality, that the roll-up starts at the Bottom level, that is, at $dom(D_c)$. 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* $\mathsf{Roll\text{-}Up}(\mathsf{Star}(C), \{\#D_c\}, D_c.(Bottom \to \ell_c); \#\mathsf{e}_\mu; \mu, F)$, *where $\#D_c$ is the unique node type in $\mathsf{Star}(C)$ that contains information on the dimension D_c and where $\#\mathsf{e}_\mu$ is the unique edge type that contains measure information on μ.*

Let $(a_1, ..., a_{c-1}, a_{c+1}, ..., a_d)$ be an element of $dom(D_1) \times \cdots dom(D_{c-1}) \times dom(D_{c+1}) \times \cdots \times dom(D_d)$ and suppose that there are r values $a_{c,i}$ from $dom(D_c)$ (for $i = 1, ..., r$) such that $(a_1, ..., a_{c-1}, a_{c,i}, a_{c+1}, ..., a_d; m_i)$ appear in the cube C, and such that all $a_{c,i}$ roll-up to the same element, call it a_{ru}, that means $\rho_{D_c.Bottom_k \to \ell_c}(a) = a_{ru}$. The roll-up on C will replace these r cells by one "new" cell which has coordinates $(a_1, ..., a_{c-1}, a_{ru}, a_{c+1}, ..., a_d)$ in $dom(D_1) \times \cdots dom(D_{c-1}) \times dom(D_c.\ell_c) \times dom(D_{c+1}) \times \cdots \times dom(D_d)$, and *which contains the aggregated measure $F(\{m_1, ..., m_r\})$. 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:*

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

To see the correctness of this claim, the substeps in the above graphoid roll-up are analysed. Firstly, $\mathsf{Climb}(\mathsf{Star}(C), \#D_c, D_c.(Bottom \to \ell_c))$ *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 $\#D_c$, 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,* $\mathsf{Aggr}(G_2, \#e_\mu, \mu, F)$ *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. So, there is nothing further to prove here.*

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 in which 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.*

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 then the proof of the roll-up case holds. The difference is that, in this case, all nodes representing D_s will contain the value all. Then, 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,* $\mathsf{n\text{-}Delete}(G_1, \#D_s)$ *is needed to delete these nodes.*

Dice. *Intuitively, the Dice operation selects the cells in a cube C that satisfy a Boolean condition φ. The syntax for this operation is* $\mathsf{Dice}(C, \varphi)$*, where φ is a Boolean condition over level values and measures. The resulting cube has the same dimensionality as the original cube. For a formal definition, refer to [16, 17]. 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 $c \in dom(D.\ell)$; or $\mu < c$, $\mu = c$ or $\mu > c$, where μ is a measure and c belongs to the domain of that measure.

Let $(a_1, ..., a_d; c_1, ..., c_m) \in dom(D_1) \times \cdots \times dom(D_d) \to dom(\mu_1) \times \cdots \times dom(\mu_m)$ be a cell of C that satisfies φ. Denote this by $(a_1, ..., a_d; c_1, ..., c_m) \models \varphi$. 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 $i = 1, ..., d$), for $j = 1, ..., m$, also satisfy φ. From $(a_1, ..., a_d; c_1, ..., c_m) \models \varphi$ it follows that there exists a k such that for all l, $(a_1, ..., a_d; c_1, ..., c_m) \models \varphi_{kl}$ 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 $(a_1, ..., a_d; c_1, ..., c_m) \not\models \varphi_{kl}$, we have $e_j \not\models \varphi_{kl}$.

This shows that exactly the edges (labelled $[\#\mu_j, c_j]$) corresponding to cells $(a_1, ..., a_d; c_1, ..., c_m)$, 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 scenarios: the classic OLAP, for which the relational model is normally used, and a *Graph OLAP* scenario, where graph metrics are aggregated. The hypothesis to be tested here is that, although the relational OLAP alternative works well 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 to perform what will be denoted here on as Graph OLAP operations.

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 Figure 1, with small changes that will be explained below. Facts are similar to the ones in Figure 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 will be compared against the relational alternative containing exactly the same data. First, two alternative relational OLAP representations are implemented on a Postgres 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.[15]

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: Timestamps of the initial and final instants of the call;
- Duration: Attribute precomputed as (StartTime - EndTime).

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

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 $(\mathit{1}, Ph_1, Ph_1, ...)$ are 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.

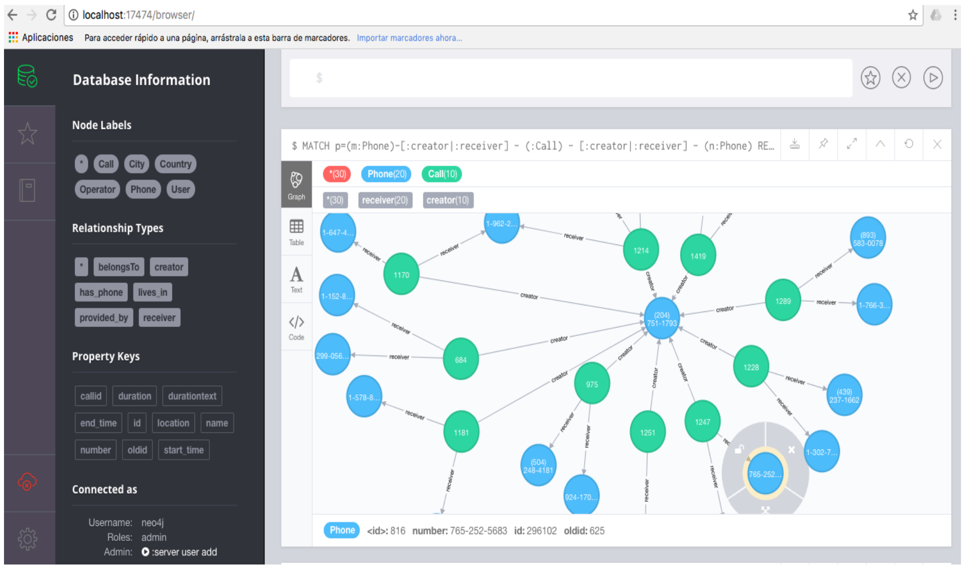
As expressed above, the background dimensions are the same of Figure 1. There are two slight differences, however, for practical reasons. First, for the Time dimension, the bottom level will have granularity TIMESTAMP, since the StartTime and EndTime attributes in the fact tables have that granularity. That means, a new level is added to the dimension. Second, in the Phone dimension the bottom level will be the phone identifier, denoted Id, which rolls up to the line number, denoted Number. This is because caller and 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 will be 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 Figure 3. Figure 12 shows a portion of the running example implemented in Neo4j. 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.

6.3 Datasets

For the relational representation, synthetic datasets of two different sizes are generated and loaded into a Postgres 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 (obviously the same in both versions), and the fourth column tells the number of tuples in the Phone dimension table.



```
$ MATCH p=(m:Phone)-[:creator|:receiver] - (:Call) - [:creator|:receiver] - (n:Phone) RE…
```

```
Phone <id>: 816 number: 765-252-5683 id: 296102 oldid: 625
```

Figure 12: Portion of the Call-graph.

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

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 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

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

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.*

The query looks for 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). For the sake of space only some of the queries are depicted. Figure 13 shows the recursive SQL query for the first representation alternative. The Cypher query for $N = 2$ is shown in Figure 14, and Figure 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 below, analyses a roll-up to the level Operator, which has less instance members than the level User.

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

Query 4 is aimed 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).

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

```
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;
```

Figure 13: Query 1 - Calls representation.

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. Although it is obviously a query appropriate for a graph representation, the SQL solution was also computed, since we are comparing the appropriateness of both representations for different kinds of queries. 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,*

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

Figure 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)
```

Figure 15: Query 1 - Cypher query for $N = 3$

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 250GB 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.

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 many cases. In Query 1, which is an aggregation over the fact graph, the relational alternatives work better.[16]

[16]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 Postgres databases have been tuned to perform in the

```
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)
```

Figure 16: Query 2 - Cypher for $N = 3$.

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

Figure 17: Query 3 - Cypher expression.

Table 3: Experimental results (running times in seconds).

Dataset	Calls $N = 2$	Calls $N = 3$	Calls $N = 4$	Calls-alt $N = 2$	Calls-alt $N = 3$	Calls-alt $N = 4$	Neo4j $N = 2$	Neo4j $N = 3$	Neo4j $N = 4$
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	11.8	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

However, when 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. However, it can be observed

best possible way. In this sense, Neo4j's performance for typical OLAP queries is, in some sense, penalized.

that when at least one end of the path is fixed in some way (eg., Query 7), the graph alternative ends in a reasonable time, while the relational ones are still unacceptable. 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 presents 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, is formally defined over graphoids. It is 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 is 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. The authors also want to thank T. Colloca, S. Ocamica, J. Perez Bodean, and N. Castaño, for their collaboration in the data preparation for the experiments.

References

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

[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):68:1–68:40, 2017.

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

[4] S-M-R. Beheshti, B. Benatallah, H. R. Motahari Nezhad, and M. Allahbakhsh. A framework and a language for on-line analytical processing on graphs. In *Web Information Systems Engineering - WISE 2012*, volume 7651 of *Lecture Notes in Computer Science*, pages 213–227. Springer, 2012.

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

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

[7] Alfredo Cuzzocrea, Ladjel Bellatreche, and Il-Yeol Song. Data Warehousing and OLAP over Big Data: Current Challenges and Future Research Directions. In *Proceedings of the Sixteenth International Workshop on Data Warehousing and OLAP*, DOLAP '13, pages 67–70, New York, NY, USA, 2013. ACM.

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

[9] L. Etcheverry and A.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*, pages 92–105, Poitiers, France, 2015.

[11] Leticia I. Gómez, Bart Kuijpers, and Alejandro A. Vaisman. Performing OLAP over graph data: Query language, implementation, and a case study. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics, BIRTE, Munich, Germany, August 28, 2017*, pages 6:1–6:8, 2017.

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

[13] Huahai He and AmbujK. Singh. Query language and access methods for graph databases. In *Managing and Mining Graph Data*, volume 40 of *Advances in Database Systems*, pages 125–160. Springer US, 2010.

[14] Ralph Kimball. *The Data Warehouse Toolkit.* J. Wiley and Sons, 1996.

[15] 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):47:1–47:15, 2015.

[16] B. Kuijpers and A. A. Vaisman. A formal algebra for OLAP. *CoRR*, abs/1609.05020, 2016.

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

[18] Qiang Qu, Feida Zhu, Xifeng Yan, Jiawei Han, Philip S. Yu, and Hongyan Li. Efficient topological OLAP on information networks. In *Proceedings of the 16th International Conference on Database Systems for Advanced Applications - Volume Part I*, DASFAA'11, pages 389–403. Springer, 2011.

[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*, pages 1139–1146, Niagara, ON, Canada, 2013.

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

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

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

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

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

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

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