An Analytics-Aware Conceptual Model For Evolving Graphs

Amine Ghrab[1,2], Sabri Skhiri[1], Salim Jouili[1], and Esteban Zimányi[2]

[1] Eura Nova R&D, Mont-Saint-Guibert, Belgium
firstname.lastname@euranova.eu
[2] Université Libre de Bruxelles, Belgium
ezimanyi@ulb.ac.be

Abstract. Graphs are ubiquitous data structures commonly used to represent highly connected data. Many real-world applications, such as social and biological networks, are modeled as graphs. To answer the surge for graph data management, many graph database solutions were developed. These databases are commonly classified as NoSQL graph databases, and they provide better support for graph data management than their relational counterparts. However, each of these databases implement their own operational graph data model, which differ among the products. Further, there is no commonly agreed conceptual model for graph databases.
In this paper, we introduce a novel conceptual model for graph databases. The aim of our model is to provide analysts with a set of simple, well-defined, and adaptable conceptual components to perform rich analysis tasks. These components take into account the evolving aspect of the graph. Our model is analytics-oriented, flexible and incremental, enabling analysis over evolving graph data. The proposed model provides a typing mechanism for the underlying graph, and formally defines the minimal set of data structures and operators needed to analyze the graph.

1 Introduction

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

NoSQL databases can be divided into four families, namely key/value stores, column stores, document databases, and graph databases. Of particular relevance to this paper is the analysis of graph databases. Graphs have the benefit

of revealing valuable insights from both the network structure and the data embedded within the structure [1]. Complex real-world problems, such as intelligent transportation as well as social and biological network analysis, could be abstracted and solved using graphs structures and algorithms. In this paper we introduce a new graph modeling approach for effective analysis of evolving graphs. By evolving we mean the variation of the values of an attribute across a discrete domain. Evolution could be over time, quantity, region, etc. In the corresponding non-evolving graph, the information would be discarded when the attributes or the topology changes.

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

We consider as a running example of this paper the Epinion product rating network [2], shown on Figure 1. The network is composed of a set of users grouped by group, and products grouped by category. Each user has a profile, a list of products ratings, and a linked by trust relationships with other users.

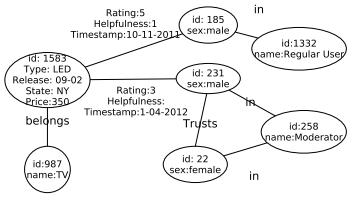


Fig. 1. Product rating network

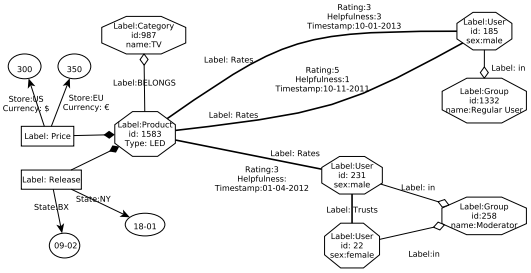


Fig. 2. Evolving product rating network

This network is sufficient to answer queries about the average rating of a product, or detection of communities of users. However, information about the evolution of the price by region or the history of rating of a given product by a user is impossible to obtain. This data is discarded and not versioned for further reuse. Hence, we enrich the original model with a typing system supporting network evolution. The evolving network keeps track of information such as the evolution of the price by region and the history of a product's rating by a user. Figure 2 depicts a part of the evolving network example. The evolving network could be used to answer rich queries like : (1) correlations between sales decrease and product rating evolution, (2) detection of popular and trendy products, and

(3) discovery of active and passive users. The previous queries could be then reused in richer scenarios such as (4) recommendation of new products, and (5) targeted advertising for influential people in the network.

The contributions of our work are summarized as follows:

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

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

2 Evolving Graph Model

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

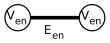
We first define the typing mechanism that characterizes nodes and edges. We propose three types of nodes, defined next.

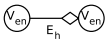
Definition 1 An **entity node** is defined by a tuple $\langle label, K_a, O_a \rangle$ where (1) *label* denotes the type of the entity node, such as user or product, (2) K_a is the map of key attributes that univocally identify the entity node, and (3) O_a is the map of optional attributes. The attributes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

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

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

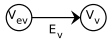
We adopt the UML notation for relationships to represent the edges of the graph. With regards to the nodes they link, we classify the edges as follows. Edges linking entity nodes are of two types:

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

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

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

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

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

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

Definition 8 An **analytics hypernode** is an induced subgraph[3,4] grouping an entity node, all its evolving and value nodes, and all edges between them. An analytics hypernode whose entity node is v is denoted as $\Gamma_v = (V, E)$, where $V \subseteq (V_{en} \cup V_{ev} \cup V_v)$ and $E \subseteq (E_{ev} \cup E_v)$. Each node (resp., edge) is part of only one hypernode: $\forall u \in V$ (resp., $e \in E$), $\exists! \Gamma_v \mid u \in \Gamma_v$ (resp., $e \in \Gamma_v$). □

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

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

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

- $\mathcal{V} = \{V_{en}, V_{ev}, V_v\}$ is the set of nodes.
- $\mathcal{E} = \{E_{en}, E_h, E_{ev}, E_v\}$ is the set of edges.
- $\alpha : (V_{en} \cup V_{ev}) \longrightarrow L_V$ is the function that returns the label for each entity or evolving node, where L_V is the set of labels of entity and evolving nodes.

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

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

– $\beta : (E_{en} \cup E_h) \longrightarrow L_E$ is the function that returns the label for each entity or hierarchical edge. L_E is the set of labels of entity and hierarchical edges.
– $\Lambda_{key} : (V_{en} \cup V_v) \longrightarrow Dom(value)$ is the function that returns the value of an attribute given its *key*. Λ is applied only to entity and value nodes. $Dom(value)$ denotes the domain of *value*.
– $\lambda_{key} : (E_{en} \cup E_h) \longrightarrow Dom(value)$ is the function that returns the value of an attribute given its *key*. λ is applied only to entity and hierarchical edges. $Dom(value)$ denotes the domain of *value*. □

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

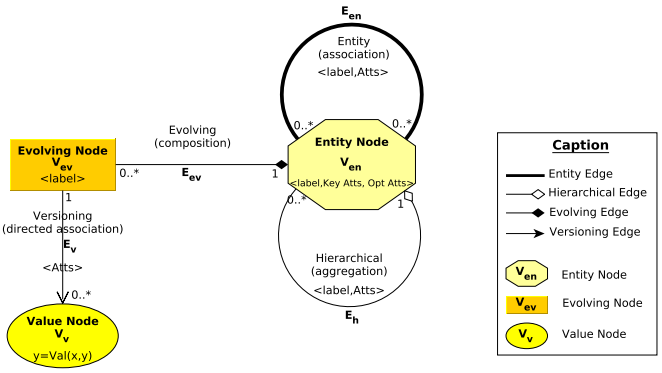


Fig. 3. Analytics-oriented evolving graph metamodel

3 Querying the Graph Model

Selection and projection are two fundamental operators in relational algebra, used to extract a subset of data according to predefined conditions on the data tuples. As their names imply, selection selects the set of tuples of a relation according to a condition on the values of their elements. Projection alters the structure of the relation by removing a subset of the elements of the tuples, and could be used to add elements by combining existing elements and constant values. In this paper, we redefine these two operators for evolving graph analysis. Then, we go a step further by introducing the traversal operation that is essential for graph analysis and provides a finer control of data extraction. However, we do not cover binary operations such as union and intersection of subgraphs, which we consider out of the scope of the model definition.

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

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

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

Definition 11 A **selection** $\sigma_{(\text{[NLabel,AttVals]*; [ELabel,AttVals]*})}(\mathcal{G})$ is a *partial*[5] subgraph extraction operation. It is applied on analytics hypernodes and the edges linking their entity nodes. It takes as input a list of the labels ($NLabel$) of the entity nodes V_{en}, underlying the targeted analytics hypernodes, (resp., a list of labels ($ELabel$) of their edges E_{en} and E_h) and the corresponding values of their targeted attributes $AttVals$. A selection returns a partial subgraph $\mathcal{G}'$ of $\mathcal{G}$ where :

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

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

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

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

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

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

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

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

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

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

In relational databases, the join operation introduces a heavy workload especially for highly connected tables [3]. In graphs, data is embedded within nodes connected through edges. The cost of running traversals within graphs is much lower than the equivalent joins in relational tables [4]. This makes graphs more suitable for highly connected data compared to relational tables. Moreover, as explained in Section 5, many of current graph databases provide partial or full support ACID properties.

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

4 Multidimensional Graph Analysis

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

We limit the study to the structures and a subset of the operators defined in the reference algebra described in [5]. However, further research is needed to device new operators uniquely useful in evolving graphs. OLAP analysis enables us to discover hidden facts and relationships between users and products, such as user satisfaction, evolution of trendy categories, and influential groups.

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

4.1 Data Structures

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

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

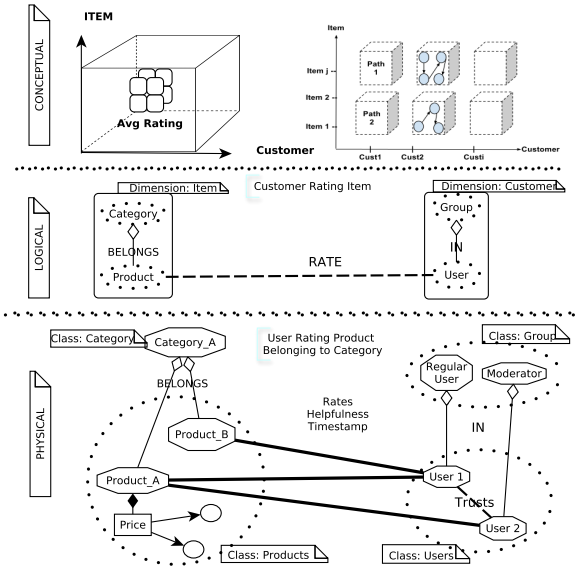


Fig. 4. Multidimensional analysis of an evolving network

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

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

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

4.2 Operations

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

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

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

Roll-Up Aggregates classes sharing a common hierarchy using the hierarchical edges. This produces a summary graph with new nodes and edges that are not necessarily present in the original graph. Aggregations could be asynchronous. We could for example study relationships between Category and User rather than Category and Group. The roll-up operators performs structural changes to the graph. If the attributes of the elements involved in the aggregation are additive, an overlay is performed and the attributes values are simply incremented. Otherwise, graph summarization techniques such as those discussed on [6–8] could be used to implement the roll-up operation.

5 Related Work

Graph analytics are gaining a lot of momentum in the data management community in recent years. A survey of graph database models according to their data model, data manipulation operators, and integrity constraints is given in [1]. Current graph databases implement different general purpose data models, without a commonly agreed conceptual modeling approach. Neo4j[6] is a centralized graph database implementing the property graph[7] model and guaranteeing ACID constraints. Titan[8] is a distributed graph database implementing property graphs and supporting ACID and partial consistency. Graph querying is made either using traversal-oriented languages such as Gremlin[9], SQL-like languages such as Cypher, or through the database core API. Our model could be implemented using any graph database that supports the input graph described on Section 2. RDF is a widespread data model in the Web community, and could be an implementation candidate for our conceptual model. Pregel [9], and its open source implementation Giraph[10], are BSP graph processing frameworks designed to execute efficiently graph algorithms. Ren et al. [10] proposed an approach to compute graph-specific measures such as shortest path and centrality within a graph

[6] `http://neo4j.org/`
[7] `https://github.com/tinkerpop/blueprints/wikiproperty-graph-model`
[8] `http://thinkaurelius.github.com/titan/`
[9] `https://github.com/tinkerpop/gremlin/wiki`
[10] `http://giraph.apache.org/`

with gradually changing edge sets. In [11], the authors present a distributed graph database system for storing and retrieving the state of the graph at specific time points. Both of these two papers are based on the redundancy offered by historical graphs trace. The analysis tasks are limited to graphs specific measures and indices with no querying or multidimensional view of data. Moreover, we consider the historical variation as a specific case of graph evolution scenarios. Related research on versioning was done by the database community. In [12], the authors suggested a conceptual model for evolving object-oriented databases by studying the evolution of objects values, schema and relationships between the objects. Although some concepts are similar, modeling the versioning depends on the data structures specific for each data model. Multidimensional analysis of graphs data has been first proposed in [8]. The authors introduce informational and topological dimensions. Informational aggregations consist of edge-centric snapshot overlaying and topological aggregations consist of merging nodes and edges by navigating through the nodes hierarchy. However, the analysis is limited to homogeneous graphs. GraphCube [7] is applied in single large centralized weighted graph and do not address different edges attributes. Yin et al. [13] introduced a data warehousing model for heterogeneous graphs. They enriched the informational and topological dimensions with the Entity dimension and the Rotate and Stretch operations along with the notion of metapath to extract subgraphs based on edges traversals. However, HMGraph did not provide semantics of OLAP operations on the proposed graph data model. Distributed processing frameworks such as Hive [14] propose data warehousing on top of large volume of data. However, they are considering only the relational model.

6 Conclusions and Future Work

In this paper, we designed a conceptual model for evolving graphs. A plethora of graph database tools is currently developed with multiple management features. However, they do not address the management of evolving networks. Moreover, no common conceptual model for efficient analysis of large evolving networks is agreed. We have proposed our contribution to evolving graph analysis by introducing a well defined conceptual model. We illustrated the model with an application on the multidimensional analysis. However, large networks analysis requires more work to build a complete stack of analysis framework. As future work we plan to proceed in warehousing the evolving graphs. Further fundamental operations such as graph aggregations should be investigated for the evolving graphs. A framework for graph data warehousing should integrate an ETL module, which takes care of matching and merging tasks and provides a graph compliant to the proposed model. An exhaustive study of new OLAP operators in evolving graphs is needed. Current graph querying languages such as Cypher should be extended to support multidimensional queries in an MDX-like fashion. Moreover, distributed processing frameworks should be integrated for large graphs processing.

Acknowledgment

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

References

1. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Comput. Surv. **40**(1) (2008) 1:1–1:39
2. Tang, J., Liu, H., Gao, H., Das Sarmas, A.: eTrust: understanding trust evolution in an online world. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM (2012) 253–261
3. Vicknair, C., Macias, M., Zhao, Z., Nan, X., Chen, Y., Wilkins, D.: A comparison of a graph database and a relational database: a data provenance perspective. In: Proceedings of the 48th Annual Southeast Regional Conference, ACM (2010) 42:1–42:6
4. Sadalage, P.J., Fowler, M.: NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional (2012)
5. Romero, O., Abelló, A.: On the need of a reference algebra for OLAP. In: Proceedings of the 9th International Conference on Data Warehousing and Knowledge Discovery, Springer (2007) 99–110
6. Tian, Y., Hankins, R.A., Patel, J.M.: Efficient aggregation for graph summarization. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, ACM (2008) 567–580
7. Zhao, P., Li, X., Xin, D., Han, J.: Graph cube: on warehousing and OLAP multidimensional networks. In: Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, ACM (2011) 853–864
8. Chen, C., Yan, X., Zhu, F., Han, J., Yu, P.S.: Graph OLAP: a multi-dimensional framework for graph data analysis. Knowl. Inf. Syst. **21**(1) (2009) 41–63
9. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, ACM (2010) 135–146
10. Ren, C., Lo, E., Kao, B., Zhu, X., Cheng, R.: On querying historical evolving graph sequences. Proceedings of the VLDB Endowment **4**(11) (2011) 726–737
11. Khurana, U., Deshpande, A.: Efficient snapshot retrieval over historical graph data. arXiv preprint arXiv:1207.5777 (2012)
12. Andonoff, E., Hubert, G., Parc, A., Zurfluh, G.: Modelling inheritance, composition and relationship links between objects, object versions and class versions. In Iivari, J., Lyytinen, K., Rossi, M., eds.: Advanced Information Systems Engineering. Volume 932 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1995) 96–111
13. Yin, M., Wu, B., Zeng, Z.: HMGraph OLAP: a novel framework for multi-dimensional heterogeneous network analysis. In: Proceedings of the 15th international workshop on Data warehousing and OLAP, ACM (2012) 137–144
14. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., Murthy, R.: Hive: a warehousing solution over a map-reduce framework. Proceedings of the VLDB Endowment **2**(2) (2009) 1626–1629