A View Handler for Semantic Graphs

Jaudete Daltio
Institute of Computing - UNICAMP
Campinas - SP, Brazil
Email: jaudete@ic.unicamp.br

Claudia M. Bauzer Medeiros
Institute of Computing - UNICAMP
Campinas - SP, Brazil
Email: cmbm@ic.unicamp.br

***Abstract*—Scientific data often come from networks with complex relationships between their entities and can be properly modeled as semantic graphs. However, once designed, there is no simple way to cross through different designs in graph databases. The goal of this research is to specify and implement a framework to overcome these limitations, allowing users to build and explore arbitrary perspectives in graphs. The framework uses the concept of views to represent a perspective. The main contribution is to help scientists run models and analyze network (graph) data according to their specific design needs. The framework is under implementation and validation using a case study on water resource data.**

I. Introduction and Motivation

Graph databases are the most suitable approach to deal with datasets where data and the relations among them have the same importance level [1], handling data networks with complex relationships between their entities [2]. A graph data model adopts variations on the mathematical definition of a graph and is relationship driven (i.e., does not require expensive operations to infer connections between data items). The most common graph data model adopted is the semantic graph. A *semantic graph* is a heterogeneous network in which every vertex and edge is typed and attributed [3]. The power of this data model lies in that semantic information resides on types and attributes [4].

The motivation for our research comes from our experience in designing semantic graphs. Usually, graph design is query-driven. The problem appears when it is necessary to provide multiple perspectives of the same semantic graph to meet different ends. There is no simple way to cross through different designs in graph databases. Ultimately, the current approach would require multiple graphs to be designed.

This challenge is the topic of our research [5], namely, to construct a framework to build and explore arbitrary *perspectives* for semantic graphs. The framework assumes graph databases as the storage model and offers operators to allow users to "see" a graph otherwise. The hypothesis that underlies this research is that the concept of view – from relational databases – is suitable to represent a perspective and that this concept can be adapted and extended to graph databases. Each view specification has its specific design and its mapping to the underlying graph database elements. Once a view is built, it can be used in network-driven analysis and exploration. A case study about Brazilian Water Resources is also under implementation to validate our framework. Water resources are usually represented as drainage networks – their physical elements being drainage points connected by drainage stretches. There are many logical elements that can be extracted from this database and each of these elements can be handled as a view over the drainage network.

This work will contribute to help scientists run models and analyze networked (graph) data according to their needs, by presenting a framework characterized by: (i) the use of graph databases as persistence model for large and heterogeneous datasets; (ii) providing different designs over semantic graphs; and (iii) support of network-driven analysis according to users' needs. To produce these results, we propose a new way of formalizing operations on graphs, which is inspired by relational database theory.

II. Theoretical Foundations and Related Work

A. Graph Data Management Paradigm

The graph data management paradigm is defined by the use of graphs as data models and the use of graph-based operations to express data manipulation [1]. A graph data model is relationship driven [6]. In this model, queries are performed through graph traversals, pattern matching or graph algorithms.

The formal foundation of graph data models is based on variations on the mathematical definition of a graph: G is a data structure composed by a pair (V, E), where V is a finite nonempty set of vertices and E is a finite set of edges connecting pairs of vertices. On top of this basic layer, several graph data structures were proposed, attempting to improve expressiveness. The most popular data model is the semantic graph (also called property graph) [7], which tries to arrange vertex and edge features in a flexible structure through types and key-value pairs.

Although graph databases have no explicit schema, we claim there is an *implicit schema* defined by the data design, which describes the semantic organization of all modeled information – i.e., specifying entities, which relations are valid for each entity and which are not, or what kinds of attributes are relevant. There are usually many alternative ways of storing the same data and even simple network-driven knowledge discovery depends on the design [8]. The usual approach is to design a graph that tries to meet most queries. Vertices are normalized and edge types should be meaningful. Connected vertices and edges are interpreted as paths and queries are expressed as graph patterns. The challenge arises when it

is necessary to provide multiple perspectives of the same semantic graph. There is no simple way to accommodate different design decisions in a graph database.

Our solution is to apply the concept of view – adapted from relational databases – to represent a perspective on top of a graph. Each view specification has its specific design and mapping to the underlying graph database elements. The concept of view adopted in this research is presented next.

B. Views

From the relational databases perspective, a *view* can be regarded as a temporary relation against which database requests may be issued [9]. Extended to our work, a graph view is a virtual perspective that "redesigns" part of a graph database. A view can be used to achieve different purposes: (i) simplify the use, extracting a portion of a database and hiding unnecessary details not relevant to the application; (ii) provide data protection and privacy policies, blocking the access to non-authorized information; or (iii) redesign data, creating virtual units derived from arrangements and aggregations of database objects [10].

Relational views are usually built by a combination of operations applied on the underlying relations, creating alternative or composite representations of existing database objects. The sequence of operations that creates a particular view is called *view generating function* (VGF). In relational theory, these operators are those used to construct queries, e.g., selection, projection, join and aggregate functions. Each view may expose the same database in a different way. Usually, a view definition is stored in the data dictionary and only materialized when a query which involves the view is requested.

III. Semantic View Handler Framework

Our framework supports the definition of view on semantic graphs on graph databases. In general terms, our view specification has two important elements: the graph view design and the mapping between the view design and source graph database elements. These two elements are represented in the framework through a *graph view generating function* (GVGF). One important difference between relational and graph data models is the existence of more than one kind of basic element – i.e., instead of relations, graph data models have vertices and edges. Thus, the regular expression of a GVGF, defined by our research, reflects this difference:

GVGF =

$$\left(\{V^{+}_{spec}, E^{+}_{spec}\}^{*} \cup \{G_{spec}\}^{*}\right)$$

As can be seen in this expression, a GVGF has two main parts, where the $*$ notation indicates zero or more occurrences of each part. In the first part, the generating function is applied to vertex and edge sets and the $+$ notation indicates at least one occurrence of each part. There are two possible scenarios: (i) using an element of the original graph under a unary operator or (ii) create a new element (e.g., vertex or edge). In the second part, the generating function is applied in terms of defining graph pattern or graph traversal, usually starting from a central concept. The graph view is the union of all elements returned by the GVGF.

Figure 1 gives a general overview of our framework architecture. It receives the specification of a GVGF as input and provides the **graph view** – a virtual perspective of the graph database – as output. The persistence layer of the framework is composed by several graph databases and it provides management mechanisms to store view definitions.

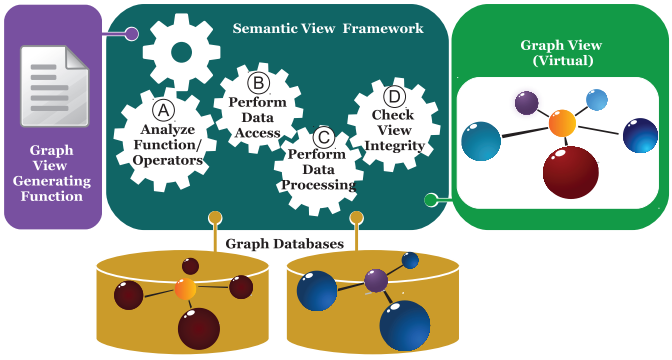

Fig. 1. Semantic View Framework Architecture

The central engine is inspired by the steps of a query processor of database systems. It is composed by four modules (following the sequence of Figure 1): **(A)** Analyze Function / Operators; **(B)** Perform Data Access; **(C)** Perform Data Processing and **(D)** Check View Integrity.

Module **(A)** parsers the generating function and builds an intermediary data structure – a tree of "plan nodes" – whose nodes represent the operators and their parameters. The framework provides an extensible set of operators that can be used to compose a view generating function. Subsequently, modules **(B)** and **(C)** work together to process each node, from the leaves to the root, accessing the required graph databases and execute each operator. These modules are executed repeatedly until the view generating function has been fully processed. Module **(D)** is responsible for ensuring that the result of the generating function is a valid semantic graph.

The first challenge addressed by our framework is to determine the graph operations that can be combined in the GVGF and specify their behavior in a high level definition. Considering a graph instead of a relation, some adaptations are necessary on classical operators: not only concerning the type of the elements (i.e., instead of relations, vertices and edges) but also dealing with edge types, vertex labels and attributes of both. We organize our graph operators as follows, combining the classical relational operators and graph operators. This organization is one of the main contributions of our research.

Projection Operator is a unary operation, represented by the symbol Π, to restrict the amount of element attributes. In relational theory, the operator is written as $\Pi_{a_1,\ldots,a_n}(R)$ where $a_1, \ldots, a_n$ is a set of attribute names of relation R. The result of such projection is the set from all tuples in R restricted

to the attributes $\{a_1, \ldots, a_n\}$. A graph projection operator is written as:

> $\Pi_{(V_{a_1,\ldots,a_n})(E_{b_1,\ldots,b_n}:t)}(V, E)$ where $a_1, \ldots, a_n$ is a set of attribute names of the vertices of V and $b_1, \ldots, b_n$ is a set of attribute names of the edges of E with type t.

Restriction Operator is a unary operator, represented by the symbol σ, to select elements which satisfy conditions. In relational theory, the operator is written as $\sigma_{\varphi}(R)$, where φ is a propositional formula to be applied on all tuples of R. The restriction selects a subset of tuples of R that satisfy the conditions φ. A graph restriction operator is written as:

> $\sigma_{\varphi,\varphi_v,\varphi_{e:t}}(V, E)$ where φ_v is a propositional formula on vertex attributes, $\varphi_{e:t}$ is a propositional formula on edge attributes with type t and φ is a path condition on vertex and edges elements – i.e., a pattern of triples vertex–edge–vertex which describes conditions over how these elements are connected.

Set Operators: refer to a category of binary operations which came from set theory: union ($\cup$), difference ($\setminus$), intersection ($\cap$) and Cartesian product ($\times$). The main difference is that, in relational theory, these operators must be applied to *compatible* relations (i.e., with the same schema). In terms of semantic graphs, it is unusual to deal with two different graphs with exactly the same "schema" – i.e., same vertex labels (i.e., representing the same real world entity), edge types, vertex and edge attributes. In this unlikely scenario, $G_a \cap G_b$ and $G_a \setminus G_b$ have the same behavior as in relational algebra. The union operator, however, can be applied on not-compatible graphs, resulting in the union of all elements involved. Cartesian product has been discarded by us, since it potentially generates too many useless semantic "garbage".

Extended Projection Operators: it refers to a broad range of operations that allow computations on the data domains. In terms of primitive domains, for example a number, we have mathematical operators. In terms of vertices or edges, we have graph properties, such as a vertex degree. These operators can be used to generate a set of values, used as attribute to existent or new elements on graph view or to generate an aggregate value, such as sum, count, average, max, min, etc.

Graph Traversal and Other Operators: The graph traversal operator allows visiting all vertices in a graph according to some criteria, updating or checking their attribute values along the way. The detailed specification of this operator and the addition of other operators are still under specification and are outside the scope of this paper.

A. Implementation Aspects

A framework prototype is under implementation. The graph database management system chosen was Neo4j [1] – a labeled property multigraph [7]. Neo4j implements a native disk-based storage manager for graphs [11]. In Neo4j, every edge must have a relationship type and there are no restrictions about the number of edges between two nodes. Both vertices and edges can have properties (key-value pairs). Neo4j offers tools, drivers and libraries, including an object-oriented API for Java, currently used in our implementation. The graph operators are being mapped to Cypher commands, according read and write query structures. Cypher is a pattern-oriented, declarative graph query language available on Neo4j.

To clarify the understanding of the framework, section IV presents a case study using real data in which different views are demanded.

[1] neo4j.com

IV. Case Study - Brazilian Water Resources Database

In Brazil, the National Water Agency (ANA) is legally responsible for ensuring the sustainable use of fresh water. To organize the required data and support management tasks, ANA constructed a water resources relational database, representing the hydrography as a drainage network composed by 620.280 drainage points and 620.279 drainage stretches. There are at least three important logical elements computed on top of this database: rivers, watersheds and main watercourses.

Several functions are routinely performed on this database, some based on the original drainage network and others based on the logical elements. Most might take advantage of the network structure. However, since the data is stored in a relational database, they are performed using recursive join operators. Our proposal is based on changing the storage solution, transforming this stored data into a stored graph, here denoted by $G_{Hydro} = (V_{dp}, E_{ds})$. All vertices in V_{dp} are labeled with **DrainagePoint** and all edges in E_{ds} have type **DrainageStretch**. Both elements keep their original attributes from the relational database. Details about the transformation process between relational and graph data model can be found in [12].

Once G_{Hydro} is created, we can apply our operators. For instance, validation tests should ensure that the drainage network is a binary tree-graph, connected and acyclic, whose edges go from the leaves to the root. The binary tree structure is checked selecting all vertices whose degree values are different from 1 (start or end points) or 3 (confluences), applying our extended projection operation. The database is inconsistent if at least one vertex is found. This task do not require generating of a specific view (i.e., they can be performed over G_{Hydro}).

A recurrent task is to determine the "most important" river in a watershed. This cannot be computed in the original hydrographic network and thus requires a graph redesign and, hence, a graph view. The logical element "river" is composed by all drainage stretches that are connected and have the same hydronym. This view is built as follows. First, it transforms each river (a set of connected edges) in one vertex. This information is generated using our projection operation: $\Pi_{distinct(hydronym)}(E : Drainage\_Stretch)$. Next, it constructs a graph in which edges represent connections between rivers. Each pair of rivers that has at least on drainage point in common are connected. An "important" river will be highly connected (i.e., a vertex with many edges) whereas less important rivers will have few connections. The graph view

result is $G_{River} = (V_r, E_c)$, in which the vertices in V_r are labeled with **River** and all edges in E_c have type **Connection**.

Although rivers are the most popular concept, the official territorial unit adopted by ANA for the management of water resources is the watershed. A watershed delimits a drainage system channel and comprises the entire area that separates different water flows. Every watershed has a main watercourse: a set of connected drainage stretches selected by performing traversal in the sub drainage network. Following the watercourse layout, the watershed can be split in a set of sub-watersheds and the process is applied recursively, as shown in Figure 2. Each watershed receives a numeric ottocode and the sub-watersheds have the same ottocode as prefix (e.g., as shown in Figure 2 watershed 454 has 9 sub-watersheds ottocoded as 4542, 4543,..., 4549). More details about this methodology can be found in [13].

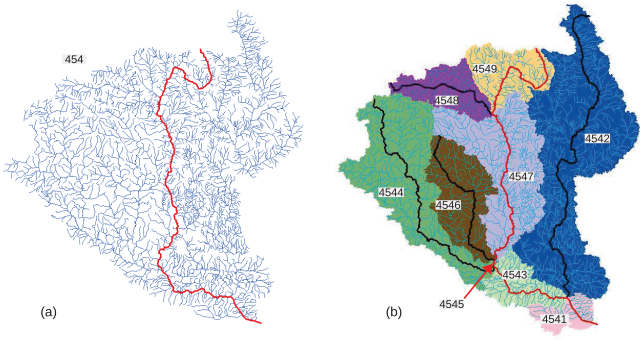

Fig. 2. Ottocoded Watersheds

An important analysis in this scenario is to identify the most influential sub-watershed of a watershed. The influence of a watershed is measured by it hydrographic catchment area upstream (from the mouth to the spring). Each drainage stretch has a hydrographic catchment area (HCA); the "upstream area" summarizes this attribute from a set of connected edges.

This analysis cannot be computed in the original hydrographic network too and thus requires a graph view. This view is built as follows. First, it transforms each watershed (a subset of the drainage network) in one vertex. This information is generated using our projection operation: $\Pi_{distinct(ottocode)}(E : Drainage\_Stretch)$. This step results in a set of vertices V_w, labeled with **Watershed**, whose vertices have an ottocode identifier.

Next, it constructs a graph in which edges represent the watershed hierarchy using our restriction operator: $\sigma_{substring(v.ottocode,length(v.ottocode-1))=(v_w.ottocode)}(V : Watershed)$. This step results in a set of edges E_h with type **PartOf** connecting only direct sub-watersheds (i.e., ottocodes that only differ on the last digit).

Finally, we have to calculate the upstream HCA of each watershed. This is done using our extended projection operator to sum the HCA of the subset drainage network. The operator is an aggregate function over a numeric attribute applied to each $v_w \in V_w$: $SUM(\sigma_{e.ottocode=v_w.ottocode}(E : Drainage\_Stretch)$.

The graph view result is $G_{Watershed} = (V_w, E_h)$ in which the influence of each watershed can be easily analyzed. It is enough to query the adjacent vertex of a given watershed and their (calculated) attributes.

V. Conclusions

This paper presented the specification of a framework to build and explore arbitrary perspectives in semantic graphs, using graph databases as the basis of data management. The approach extends the concept of views to represent a virtual graph, which redesigns the original data. The definition of a graph view is based on a graph view generated function, composed by a set of graph operators, inspired by relational algebra. To complement these operators, our framework defines other operators based on classical graph algorithms. Besides executing and to combining the results of all operators, the proposed framework check the consistency of the generated view to ensure that it is a valid graph.

The implementation and validation of the framework are in progress. The internals of the framework were explained via examples of functions performed in a real database of water resources, pointing out some of challenges faced.

Acknowledgment

Work partially financed by FAPESP/CCES (2013/08293-7), FAPESP-PRONEX (eScience project), INCT in Web Science, and individual grants from CNPq.

References

[1] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1:1–1:39, Feb. 2008.

[2] V. Dhar, "Data science and prediction," *Commun. ACM*, vol. 56, no. 12, pp. 64–73, Dec. 2013.

[3] J. F. Sow, *Conceptual Structures: Information Processing in Mind and Machine*. Reading: Addison-Wesley, 1984.

[4] M. Barthelemy, E. Chow, and T. Eliassi-Rad, "Knowledge representation issues in semantic graphs for relationship detection," in *AAAI Spring Symposium: AI Technologies for Homeland Security*, 2005, pp. 91–98.

[5] J. Daltio and C. B. Medeiros, "Handling multiple foci in graph databases," in *Lecture Notes in Bioinformatics (LNBI) - Proceedings of 10th International Conference on Data Integration in the Life Sciences*, S. I. P. Switzerland, Ed., vol. 8574, Lisboa, Portugal, 2014, pp. 58–65.

[6] T. Goodwin and S. M. Harabagiu, "Automatic generation of a qualified medical knowledge graph and its usage for retrieving patient cohorts from electronic medical records," in *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*, 2013, pp. 363–370.

[7] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. O'Reilly Media, Incorporated, 2013.

[8] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big data and its technical challenges," *Commun. ACM*, vol. 57, no. 7, pp. 86–94, Jul. 2014.

[9] A. Furtado, K. Sevcik, and C. Santos, "Permitting updates through views of databases," *Informations Systems*, vol. 4, pp. 269–283, 1979.

[10] C. B. Medeiros, M.-J. Bellosta, and G. Jomier, "Multiversion views: Constructing views in a multiversion database." *Data Knowl. Eng.*, vol. 33, no. 3, pp. 277–306, 2000.

[11] P. Kivikangas and M. Ishizuka, "Improving semantic queries by utilizing unl ontology and a graph database," in *Semantic Computing (ICSC), 2012 IEEE Sixth International Conference on*, Sept 2012, pp. 83–86.

[12] J. Daltio and C. B. Medeiros, "HydroGraph: Exploring Geographic Data in Graph Databases," in *Proc XVI GEOINFO*, 2015, pp. 44–55.

[13] O. Pfafstetter, "Classificao de bacias hidrogrficas: metodologia de codificao." Departamento Nacional de Obras de Saneamento (DNOS). Rio de Janeiro, RJ, 1989.