How can graph databases and reasoning be combined and integrated?

Edelmira Pasarella

[a] *Universitat Politecnica de Catalunya, 08034, Barcelona, Spain*

Abstract

Nowadays the graph data model has been accepted as one of the most suitable data models to formalize relationships among entities of many domains. Deductive databases based on the Datalog language have been used to deduce new information from large amounts of data. Most of the attempts to combine logic and graph databases are based on translating knowledge in graph databases into Datalog and then use its inference engine. We aim to open the discussion about combining graph databases and a graph-oriented logic to define "native" deductive graph databases. This is, graph databases equipped with an inference mechanism based on graph based logic. To be concrete, we plan to use the recently introduced graph navigational logic.

Keywords: Graph databases, deductive databases, property graphs, graph navigational logic, graph patterns queries, path queries

1. Introduction

Nowadays the graph data model is accepted as one of the most suitable data models to formalize relationships among entities of many domains. Despite it could sound like a *cliché*, it is worth mentioning that, in the strongly interconnected world we live, graphs rise everywhere. This is why in the last years there has been a lot of effort in the data management scientific community to provide theoretical foundations, query languages and tools to make the graph data model amenable for practical purposes.

In general, in a database system the main components are the *schema*, the *instance* and the *integrity constraints*. In a graph database system the instance is a data graph. Lately, the *property graph* model has been adopted as the standard model underlying graph databases [4, 6]. A property graph is a multidigraph with labeled nodes and edges. In property graphs, nodes and edges (and even paths [2, 5]) have attributes called *properties*. Properties are typed, however unlike attributes in the relational model, not all the possible properties of a node or an edge (or a path, when this applies) are forced to be defined. *Neo4j* [16] is one of the graph database management systems using the property graph model. Let us consider the following example. Suppose that an International Office for the Prevention of Drug Trafficking maintains a Drug Trafficking Crime Graph Database (DGDB). Figure 1 depicts (part of) a property graph.

Based on [2, 8], the schema is a (meta) graph that specifies the structure of the data graph, i.e. the types of nodes, the type of edges, the attributes (of both, nodes and edges) a their datatypes.

*This work is partially supported by MCIN/ AEI /10.13039/501100011033 under grant PID2020-112581GB-C21.

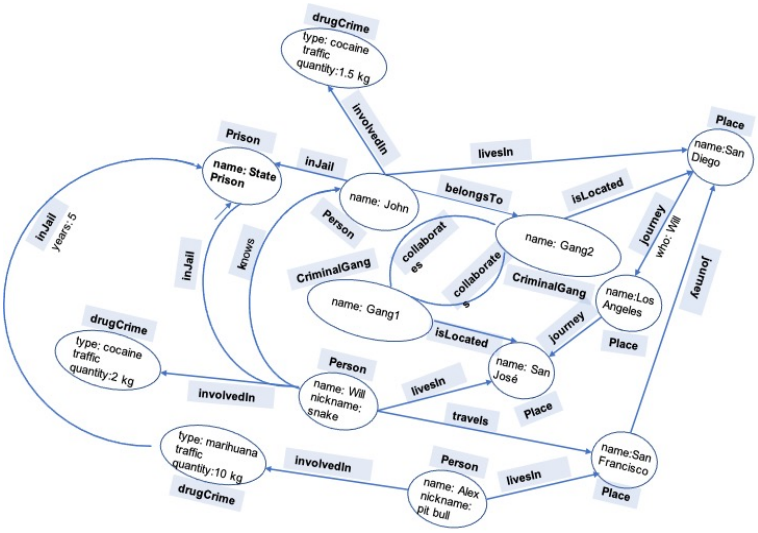

Figure 1: **Partial view of the DGDB instance**. Bold names in shadow boxes are nodes and edges labels, respectively. As we have said previously, not necessarily all the properties are defined in a property graph. In this case, edges properties years and who are defined for some inJail and journey edges. Node properties name, nickname, quantity and type are defined for some Person, drugCrime, Prison, Place and criminalGang nodes.

This is, nodes in the schema graph represent classes of nodes in the *instance*. Edges in the schema graph represent classes of links between nodes in the *instance*. The schema also includes hierarchical relationships, if any. Moreover, one can consider that the database is parameterized by a *type support schema*, $\mathcal{X}$, where data types are defined in terms of their sorts and binary relations over these sorts. This parameter is analogous to the *attribute support schema* defined in [15] In the case of the Prevention of Drug Trafficking Crime Graph Database, $\mathcal{X} = \langle \mathsf{DT}, \mathsf{NT}, \mathsf{ET}, \mathsf{NP}, \mathsf{EP}, \mathsf{NPT}, \mathsf{EPT} \rangle$, where

$$
\begin{array}{ll}
\mathsf{DT} = & \{\mathsf{Int}, \mathsf{String}, \mathsf{Float}, \mathsf{Date}\}, \text{ where} \\
\mathsf{Int} = & (\mathsf{integer}, \{=_{\mathsf{int}}, \neq_{\mathsf{int}}, <_{\mathsf{int}}, <_{\mathsf{int}}, \leq_{\mathsf{int}}, \geq_{\mathsf{int}}\}) \\
\mathsf{String} = & (\mathsf{string}, \{=_{\mathsf{string}}, \neq_{\mathsf{string}}\}) \\
\mathsf{Float} = & (\mathsf{float}, \{=_{\mathsf{float}}, \neq_{\mathsf{float}}, <_{\mathsf{float}}, <_{\mathsf{float}}, \leq_{\mathsf{float}}, \geq_{\mathsf{float}}\}) \\
\mathsf{Date} = & (\mathsf{date}, \{=_{\mathsf{date}}, \neq_{\mathsf{date}}, <_{\mathsf{date}}, <_{\mathsf{date}}, \leq_{\mathsf{date}}, \geq_{\mathsf{date}}\})
\end{array}
$$

DT is the set of sorts in the schema and the binary relations over these sorts[1].

$$
\begin{array}{ll}
\mathsf{NT} = & \{\mathsf{Person}, \mathsf{Place}, \mathsf{CriminalGang}, \mathsf{Prison}, \mathsf{drugCrime}\} \\
\mathsf{ET} = & \{\mathsf{journey}, \mathsf{nearby}, \mathsf{isLocated}, \mathsf{collaborates}, \mathsf{areRivals}, \mathsf{belongsTo}, \mathsf{knows}, \mathsf{livesIn}, \mathsf{travels}, \\
& \mathsf{inJail}, \mathsf{involvedIn}, \mathsf{isDirector}\}
\end{array}
$$

are the set of types of nodes and the set of types of edges, respectively.

$\mathsf{NPT} : \mathsf{NP} \to \mathsf{DT}$ and $\mathsf{EPT} : \mathsf{EP} \to \mathsf{DT}$ are functions that map nodes properties and edges properties to their corresponding data type. For example, $\mathsf{NPT}(\mathsf{name}) = \mathsf{String}$, $\mathsf{EPT}(\mathsf{since}) = \mathsf{Date}$ and so on.

[1]Notice that we leave out the details about the types (Date, Date) and 1..5 in the schema graph of Figure 2.

$$\mathsf{NP} = \{\mathsf{name}, \mathsf{locType}, \mathsf{country}, \mathsf{secLevel}, \mathsf{type}, \mathsf{quantity}, \mathsf{location}, \mathsf{nickName}, \mathsf{age}, \mathsf{citizenship}\}$$
$$\mathsf{EP} = \{\mathsf{date}, \mathsf{who}, \mathsf{Transport}, \mathsf{since}, \mathsf{address}, \mathsf{years}, \mathsf{month}, \mathsf{poosition}, \mathsf{status}, \}$$

are the set of properties of nodes and the set of properties of edges, respectively.

Figure 2 depicts the schema graph for the Prevention of Drug Trafficking Crime Graph Database. Graph database approaches differ in the definition of the integrity constraints. However, most

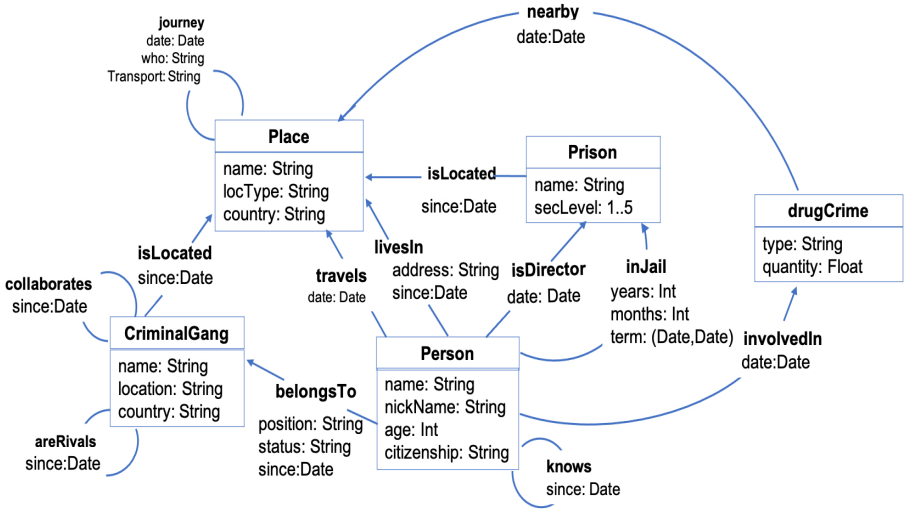

Figure 2: **Drug Trafficking Crime Graph Database Schema**. The attributes in nodes and edges are typed properties. Bold names are labels.

of them require the *schema-instance consistency* as data-modeling and integrity checking tool [3]. This is, an instance graph must satisfy the structural restrictions given by the schema graph [2]. Considering the three components of a database mentioned above, we can assume that a graph database is a pair $(\mathsf{S}_{\mathcal{X}}, \mathsf{G})$ where $\mathsf{S}_{\mathcal{X}}$ is a schema parameterized by $\mathcal{X}$ and G, the instance graph, is a data (property) graph satisfying the *schema-instance consistency* integrity constraint.

Once the data model and the schema are established and a database begins its operation from scratch, the processes of evolution of contained data and querying it take place according to the use of applications based on that database. The evolution of a (relational) database, i.e. successive changes induced by insertion, deletion and update operations, change the instances of some tables but do not modify the structure of the underlying relational model. This is, depending on the operation, after it takes place, some tables have more/less rows when an insertion/deletion takes place or some attribute values have changed according to an update. However, the number of tables and their attributes/types are invariant with respect to any of these operations. On the contrary, this is not the case with the evolution of graph databases. There are topological issues to be considered. Indeed, after inserting or deleting edges, nodes and even entire subgraphs, topology of the graph databases change, possibly in a drastic way. For instance, after deleting a node, the graph can become disconnected and after inserting an edge new paths can occur. In general, new topological graph patterns can appear and other existing prior the operation can disappear. Apart from topological issues, since nodes and edges as data containers in graph databases are not monolithic entities, update operations of graph databases could modify these attributes. Therefore, although topology remains the same, it could happen that after an update operation the minimum path between a

couple of nodes in the graph would no longer satisfy that property.

Regarding querying, according to [4] there are two natural querying operations in the context of graph databases: *graph pattern matching queries* and *path queries*. Regarding graph pattern matching queries, a pattern is a graph in which nodes and edges can be variables possibly equipped with some constraints over some node/edge properties. Roughly speaking the evaluation of a graph pattern matching query against a graph database consists in finding a mapping from variables in the graph pattern to constant values in the instance such that, when the mapping is applied to the pattern, the resulting graph is contained in this data graph. This is, the resulting graph is a subgraph of the data graph. In regards to path queries, they corresponds to querying for specific path patterns. The most general form is querying for a path pattern whose initial and ending points are connected by a regular expression. The evaluation of a navigational query against a graph database consists in computing/listing paths in the data graph such that the concatenation of the labels in their edges are words in the language of the regular expression in the path-pattern. Bonifati et al. [6] study in details the most representative kinds of path queries. The combination of path queries and pattern matching queries is called *navigational queries*.

A deductive database system [13] is a database management system based on a logical model and equipped with an inference engine and a logic query language (for instance, Datalog). In a deductive database the instance database is called *extensional database*. Additionally, in a deductive database system there is a set of logic programming style rules allows for deducting new knowledge. This set of rules is called the *intensional database*. Rules in the intensional database include predicates/relations no occurring in the extensional database. Since deductive databases based on a Datalog language [1, 7] have been used to deduce new information from large amounts of data (mostly relational databases), it is worth wondering how to deduce new knowledge based on graph databases. There are some pioneer works combining logic and graph databases. Paredaens et al. [10, 11] introduce G-Log. This approach is based on a mapping that transforms an instance over a G-Log scheme into an interpretation over the many sorted first-order language corresponding to the schema. In [12], the instance of a graph database is translated into a Datalog extensional database on relational tables. The intensional databases are Datalog rules. This implies that, instead of deducting new patterns (subgraphs), typical Datalog query answers are obtained. The same approach has been recently used in [14] to address the problem of discovering relationships over a drug-drug interactions graph database. In [6], *regular property graph logic* queries are presented as a variant of Datalog (syntax and semantics) extended for property graphs. From our point of view, a graph logic providing an inference mechanism on property graphs is the main requirement to address the problem of defining a "native" deductive graph databases. By "native" we mean an inference mechanism directly applying on graphs patterns instead of (relational) extensional Datalog database. In our pure graph approach, the knowledge (i.e. extensional database) will be the instance data/ground (property) graph and rules (intensional database) will be expressed in terms of graph patterns.

Recently, Navarro et al. [9] have defined a *graph navigational logic* that allows for formalizing property graphs, including path properties. This logic is equipped with a sound and complete deductive (tableau) method. We have insights that *graph navigational logic* and its deductive mechanism provides the foundation for "native" graph deductive databases. In this work we aim to open the discussion about this topic.

2. Motivating Example

In this section we present an example of a rule for inferring new knowledge from a Drug Trafficking Crime Deductive Graph Database. Part of the extensional database is given in Figure 1.

Example. Let us suppose that by their experience, analysts of the Office for the Prevention of Drug Trafficking, use the rule formalized in Figure 3 to prevent new drug trafficking crime networks in some places. Following a logic programming style, the left part corresponds to the head of the rule and the right part corresponds to the body. This is, there is a head graph pattern (left) and a body graph pattern (right). In the body pattern persons p1 and p2 have been involved in the same type of drug crime. Moreover, p1 and p2 have shared some time in prison pr. Double links labelled with regular expressions represent paths. The path e12 between p1 and the place t2 corresponds to, first a travel of p1 to some intermediate place and then, a journey through one or more places to t2. This is the place where p2 lives and the gang g2 is located. In addition, p2 belongs to g2 and this gang and gang g1, located in t1 cooperate with each other. This pattern gives insights of a possible new drug crime activity by person p1 in the place t1. Moreover, p1 seems to be connected to the gang g1 and, hence, this gang could start operating in t1. This is represented by the head pattern. Notice that some properties for paths, non specified in the schema, are being used in the body of the rule (e12.term and e16.term). Together the rising of new nodes and edges in the patterns of intensional graph database, the management of paths properties is one of the feature we plan address in our proposal. Roughly speaking, we envision a querying-matching mechanism as follows: If (i) the query pattern (goal) matches the head of the rule, (ii) there exists a match for the body pattern against the data graph and (iii) the property-constraints are satisfied, then the answer will be an instance of the head pattern. Otherwise, no answer is obtained.

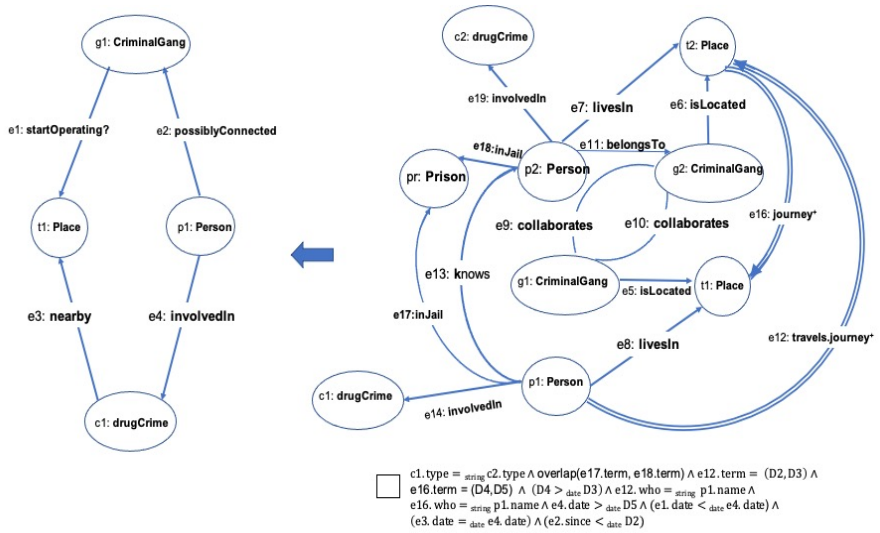

Figure 3: **Rule for discovering possible new drug trafficking crime in some place**. Following a logic programming style, the left part corresponds to the head of the rule and the right part corresponds to the body. Notice that in the head graph there are new kind of edges, **startOperating?** and **possiblyConnected**. Variables occurring in patterns are denoted by p1, p2, t2, etc. Logical expression below the body pattern, at the right of □, is a constraint with respect to some properties of nodes/edges in the patterns. This constraint is based on the relational operators and the predicates provided by the type support schema-constraint.

3. Our Proposal

The question in the title of this paper has been formulated in [17]. In that work authors pose the need of developing logic-based and declarative formalisms for graph processing. In particular,

they ask "Which underlying formalisms make this possible?". To address this problem we plan to define a constraint graph-logic formalism having the graph navigational logic [9] as underlying logic. This formalism must be amenable for representing instances (extensional databases), integrity constraints and production-rules (intensional databases). Since new entities (edges and/or nodes) and their properties could appear in rules, it will be necessary a way to consistently extend, when necessary, the type support schema-constraint. Moreover, possibly new integrity constraints may also be consistently added and the use of some forms of aggregation [4] required.

References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.

[2] R. Angles. The property graph database model. In *AMW (CEUR Workshop Proceedings), Vol. 2100. CEUR-WS.org.*, 2018.

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

[4] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, and D. Vrgoč. Foundations of modern query languages for graph databases. *ACM Computing Surveys (CSUR)*, 50(5):1–40, 2017.

[5] R. Angles, M. Arenas, P. Barceló, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda, et al. G-core: A core for future graph query languages. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1421–1432, 2018.

[6] A. Bonifati, G. Fletcher, H. Voigt, and N. Yakovets. Querying graphs. *Synthesis Lectures on Data Management*, 10(3):1–184, 2018.

[7] C. J. Date. *Relational database: selected writings*. Addison-Wesley Longman Publishing Co., Inc., 1986.

[8] H. Lbath, A. Bonifati, and R. Harmer. Schema inference for property graphs. In *EDBT 2021-24th International Conference on Extending Database Technology*, pages 499–504, 2021.

[9] M. Navarro, F. Orejas, E. Pino, and L. Lambers. A navigational logic for reasoning about graph properties. *Journal of Logical and Algebraic Methods in Programming*, 118:100616, 2021.

[10] J. Paredaens, P. Peelman, and L. Tanca. G-log: A graph-based query language. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):436–453, 1995.

[11] J. Paredaens, P. Peelman, and L. Tanca. Merging graph-based and rule-based computation: The language g-log. *Data & knowledge engineering*, 25(3):267–300, 1998.

[12] K. Rabuzin. Deductive graph database–datalog in action. In *2015 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 114–118. IEEE, 2015.

[13] R. Ramakrishnan and J. D. Ullman. A survey of deductive database systems. *The journal of logic programming*, 23(2):125–149, 1995.

[14] A. Rivas and M.-E. Vidal. Capturing knowledge about drug-drug interactions to enhance treatment effectiveness. In *Proceedings of the 11th on Knowledge Capture Conference*, pages 33–40, 2021.

[15] S. Z. R. Rizvi and P. W. Fong. Efficient authorization of graph-database queries in an attribute-supporting rebac model. *ACM Transactions on Privacy and Security (TOPS)*, 23(4):1–33, 2020.

[16] I. Robinson, J. Webber, and E. Eifrem. *Graph databases: new opportunities for connected data.* " O'Reilly Media, Inc.", 2015.

[17] S. Sakr, A. Bonifati, H. Voigt, A. Iosup, K. Ammar, R. Angles, W. Aref, M. Arenas, M. Besta, P. A. Boncz, et al. The future is big graphs: a community view on graph processing systems. *Communications of the ACM*, 64(9):62–71, 2021.