A new Nested Graph Model for Data Integration

Giacomo Bergami

2/10/2017

Introduction

- Graph data are a widely adopted.
- Currently, most of the No-SQL data representation is semistructured (JSON, XML).
- Graph operations allowing to integrate property graphs and semi-structured (**nested**) data are not provided.
  - An **intermediate data representation** allowing to represent both structured, semi-structured (nested) and non-structured data is missing.
  - A query language for this generalized data model is required. It should be able to sketch the data integration task and express both semistructured and graph queries.

Related Works: Data Integration

- *Schema Oriented*, uniform data representation
  - A schema can be always extracted if missing (**generalization**).
  - **Schema alignment** allows to find data similarities.
  - All the data source schemas shall be aligned to the final **hub schema**.
- *Query Oriented*
  - The schema alignment can be represented as a query rewriting.
  - We can perform each query separately on the original sources first and then integrate the intermediate results (**LAV**) or translate and align the data first and then perform the query (**GAV**).
  - In both scenarios, part of the query has to be performed on top of the intermediate representation.

Contributions

- An operation allowing to integrate different graphs was missing. Definition of the first class of binary operators for property graphs: **graph joins**.
  - Its definition pointed out the inefficiency of current graph query plans and data structures.
- Definition of a data model allowing to provide data integration between graph, semistructured and structured data.
  - **Future works**

Graph Joins: A query example

Consider an on-line service such as ResearchGate where researchers can {follow} each others' work, and a citation graph. Return the paper graph where **a paper cites another one iff. the first author of the first paper *follows* the first author of the second**.

- Binary operator returning just one graph.
- Vertex conditions are (θ-)join conditions.
- **A way to combine the edges is determined**.

Graph Joins: Relational Query Plan

V_{ESN}

id	NAME	ℓ_v
6	Alice	{Empl}
7	Bob	{Empl}
8	Carl	{Empl}
9	Dan	{Empl}

θ

$V_{Projects}$

id	TITLE	NAME	ℓ_v
1	t_1	Alice	{Proj}
2	t_2	Alice	{Proj}
3	t_3	Bob	{Proj}
4	t_4	Carl	{Proj}
5	t_5	Dan	{Proj}

E_{ESN}

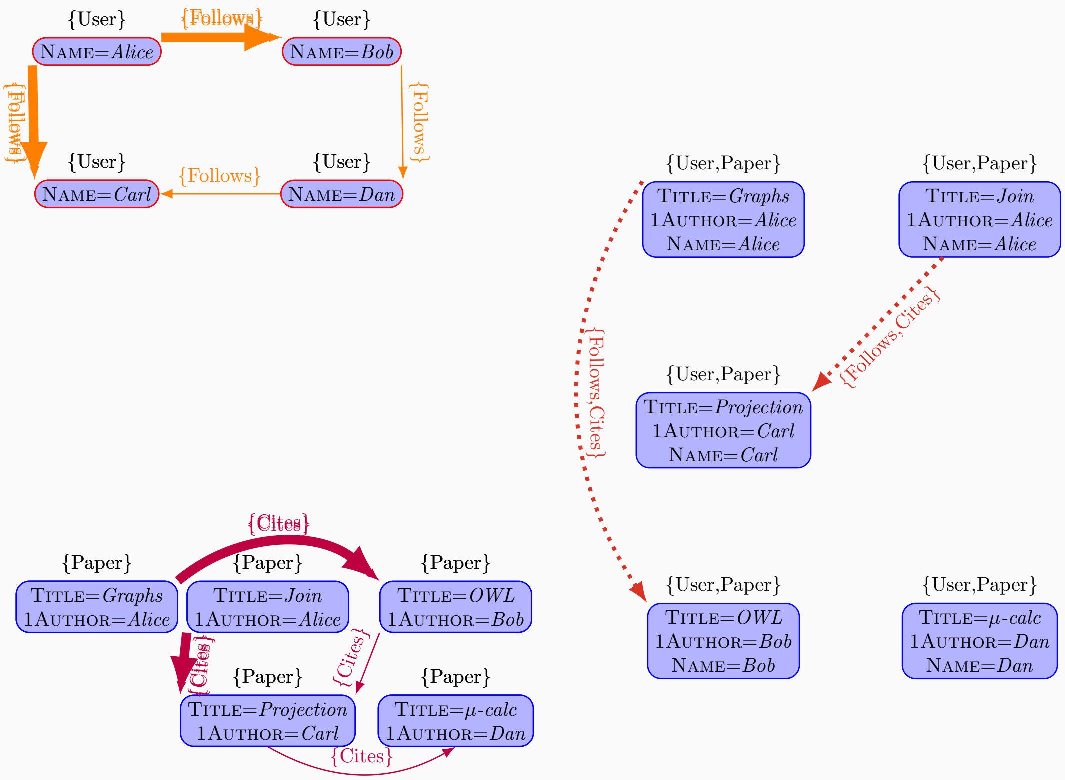
id	src	dst	ℓ_e
5	6	7	{WorksWith}
6	6	8	{WorksWith}
7	7	9	{WorksWith}
8	9	8	{WorksWith}

$E_{Projects}$

id	src	dst	ℓ_e
1	1	3	{Needs}
2	2	4	{Needs}
3	3	4	{Needs}
4	4	5	{Needs}

Figure 1: A relational representation of the graph, where the vertices are θ-joined.

Graph Joins: The query result



Graph Joins: Goals

1. The **data model** must *enhance the serialization* of both operands and graph result.
2. The **join definition** must be flexible enough to support further extensions (modularity, compositionality and properties preserving).
3. The **physical model** must allow a *quick access* to the data structures.

Graph Joins: Physical Model

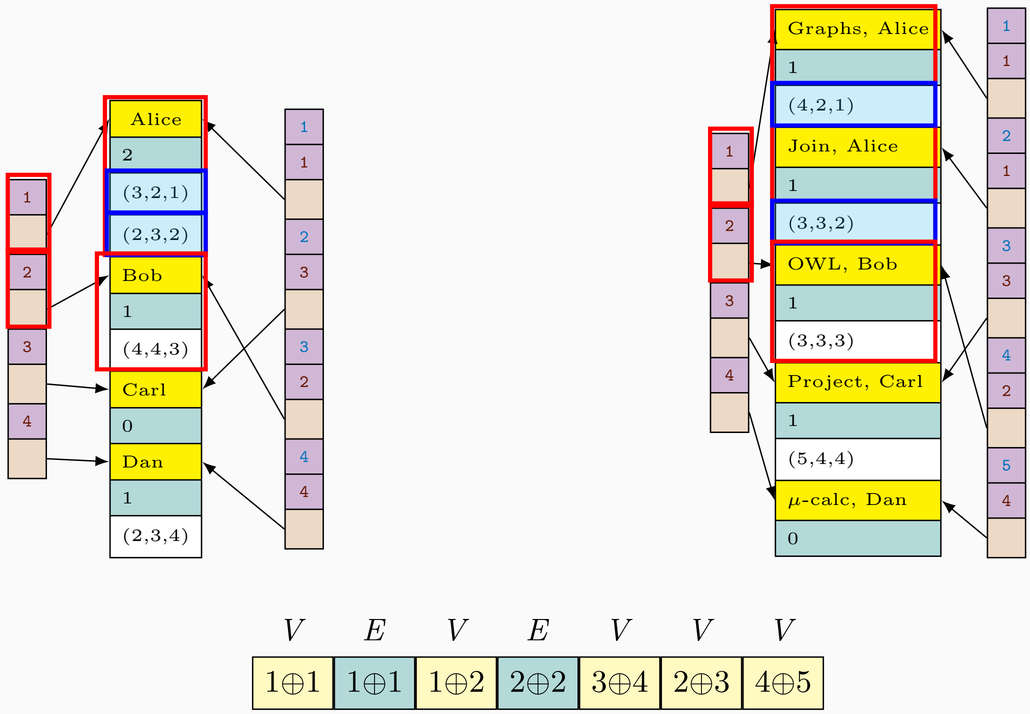


Figure 3: Representing Graph Operators and Output

Graph Joins: Benchmarks

Operands Size		**Result**		**Join Time (C/C++)** (ms)			**Join Time (Java)** (ms)	
Left ($\|V\|$)	Right ($\|V\|$)	Size ($\|V\|$)	Size ($\|E\|$)	Virtuoso	PostgreSQL	GCEA (C++)	Neo4J	GCEA (Java)
10	10	5	2	4.99	11.29	0.53	211.45	24.97
10^2	10^2	16	4	4.94	22.82	0.93	222.87	32.70
10^3	10^3	251	55	4.55	22.92	4.35	448.97	117.58
10^4	10^4	2 734	680	117 712.00	183.90	40.42	3 149.90	1 150.37
10^5	10^5	26 803	7 368	>4H	7 150.74	411.78	241 026.79	17 178.49
10^6	10^6	151 212	99 558	>4H	99 683.91	3 966.72	>4H	178 066.80

Figure 4: Conjunctive join execution on well known graph databases. Similar result were provided on graph database libraries.

Graph Joins: Limitations (1)

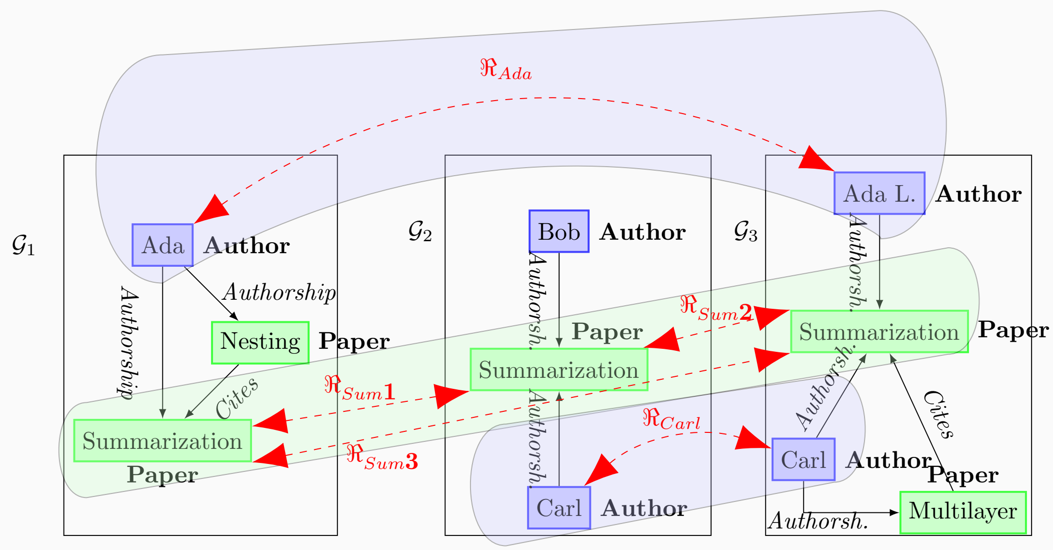


Figure 5: A data integration scenario, where the schema is already aligned

Graph Joins: Limitations (2)

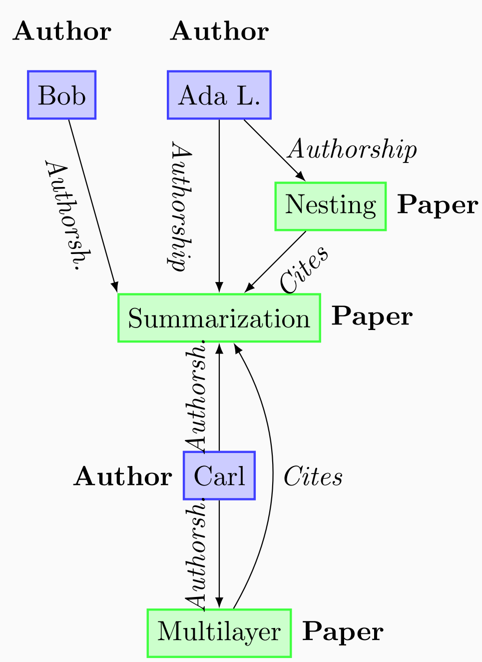


Figure 6: A data integration scenario, where the schema is already aligned

Graph Joins: Limitations (3)

- Suppose to generalize the graph join by extending the θ-join ($\ltimes_\theta$) to a full θ-join ($⟗_\theta$) over the vertices.
  - This is possible within the previous definition.
- Suppose to use as a θ some edges that remark the similarity values among the nodes.
  - After the pairwise application of the join, the θ edges must be rewritten.
- In some data cleaning scenarios, we want to preserve the original non-aggregated information, while providing an integrated view as in the previous picture.
  - **A new data model is required**, generalizing statecharts and hypernodes.

Generalized Semistructured Model and Nested Graphs

$$GSM = (o, O, \ell, \xi, \phi)$$

- o is the **reference object**, designing the root of the GSM.
- ϕ is the object id containment function for each expression e ($\phi(o', e)$).
- O is the supset of all the objects contained by o ($\bigcup_e \phi(o', e)$)
- ℓ is the function associating to each object in O a list of labels.
- ξ is the function associating to each object in O a list of expressions.

GSM, generalizing semistructured data containment

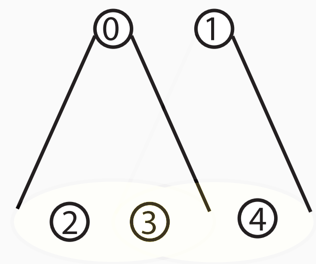


Figure 7: Allowing multiple containments of the same object.

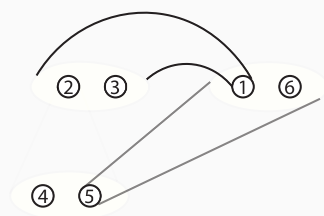


Figure 8: Allowing multiple nesting and nestings with recursions.

GSM, representing (nested) graphs

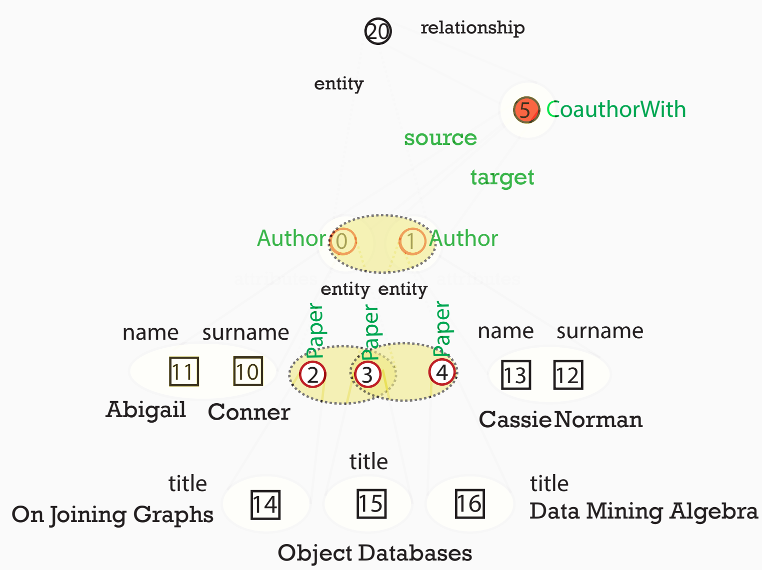


Figure 9: Representing a nested graph with two vertices and one edge.

GSM, representing semistructured documents

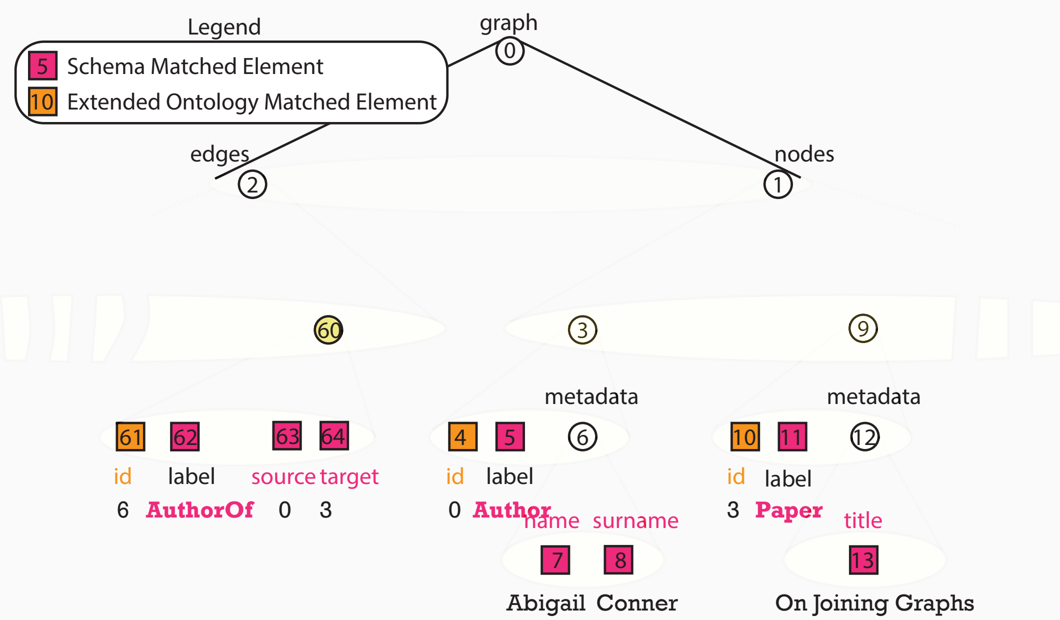


Figure 10: Another graph representation from a JSON document.

GSM, aligning graphs with semistructured schemas (1)

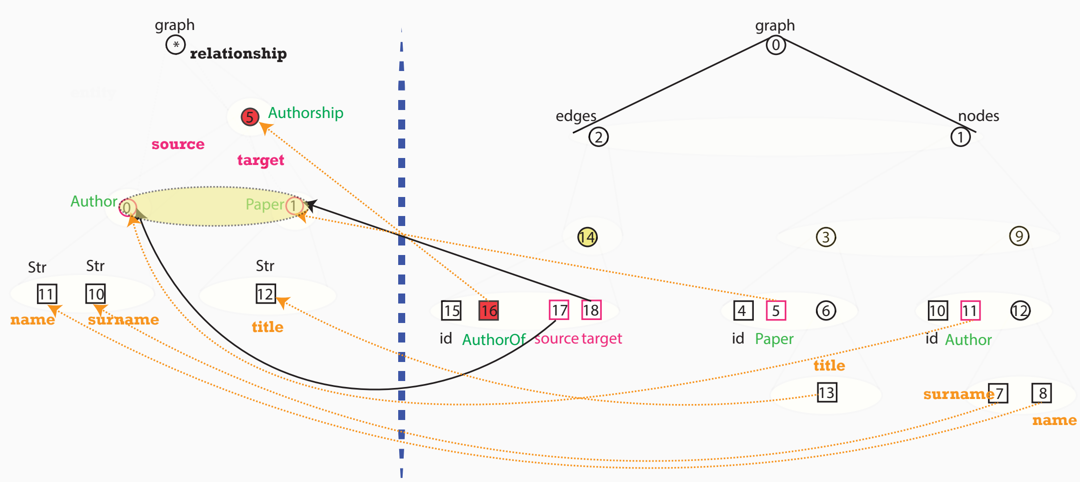


Figure 11: Schema alignment, using the usual techniques.

GSM, aligning graphs with semistructured schemas (2)

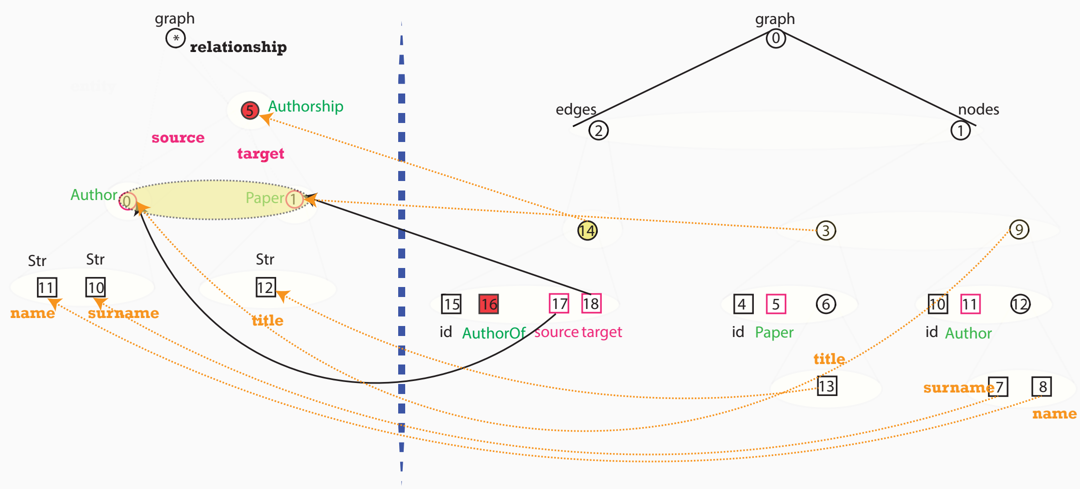


Figure 12: Schema alignment, using the nesting information refinement.

GSM, aligning graphs with semistructured schemas (3)

The previous data alignment process provides an interesting perspective for graph grammars:

- The schema (*on the right*) extracted from the original data defines the **matching graph**.
  - Given that the schema was extracted from the original data, we already know the result of such match.
- The hub schema (*on the left*) defines how the data on the right has to be rewritten **transformation graph**, using the edges as transformations.

Hereby, a generalization of the graph grammars is required. The operations required to do the matching and transformations are non trivial. Thefore, an algebra expressing such operations is required.

paNGRAm Algebra

paNGRAm	*COA Algebra*	*3W Algebra*	*Graph Languages*
Selection, σ_P	Selection, σ_P Projection, π_P	Selection, σ_P	Graph Traversal, P
Map, $\mu_{\ell',\xi',\phi'}$	Embedding, ε_ϕ Projection, π_ϕ	Calc, $Calc_f$	Transform, Pattern Matching
Creation, $\kappa^{\omega}_{L,E}$	Embedding new elements	Regionize, κ	Transform,
Fold, $\textsf{fold}_{S,f}$	Expression using ς	Loop operator, λ	Reduce (GrAlA)
Disjoint Union, $\sqcup$	$+\mu$ = Set operations	$+\mu$ = Set operations	Graph Binary Operators

- An operation for creating new elements is required because this algebra manipulates ids (already existing objects).
- All the other operations (such as graph unnesting ν, splicing ς, graph joins $\Join$) are expressible through a composition of operators.

Future Works

- In order to acomplish the data integration scenario, the alignment between the source schemas and the hub schema shall be defined.
- Algebraic expressions reproducing the single steps shall be defined.
- Check if the whole algorithmic plan can be implemented more efficiently than the single operators.