






Serialization for Property Graphs

Dominik Tomaszuk¹(✉) , Renzo Angles^{2,3} , Łukasz Szeremeta¹ ,
Karol Litman¹, and Diego Cisterna³

¹ Institute of Informatics, University of Białystok,
Ciołkowskiego 1M, 15-245 Białystok, Poland
{d.tomaszuk, l.szeremeta}@uwb.edu.pl

² Department of Computer Science, Universidad de Talca, Curicó, Chile
rangles@utalca.cl

³ Center for Semantic Web Research, Santiago, Chile
dcisterna@live.com

Abstract. Graph serialization is very important for the development of graph-oriented applications. In particular, serialization methods are fundamental in graph data management to support database exchange, benchmarking of systems, and data visualization. This paper presents YARS-PG, a data format for serializing property graphs. YARS-PG was designed to be simple, extensible and platform independent, and to support all the features provided by the current database systems based on the property graph data model.

Keywords: Serialization · Property graph · Graph database

1 Introduction

Data serialization is the process of converting data (obtained from a source system) into a format that can be stored (in the same system) or transmitted (to a target system), and reconstructed later. Data serialization methods are applied in several situations [6, 21, 27], in particular when an ETL process is required (i.e. when the data need to be extracted, transformed and loaded). Data serialization implies the definition of a data format with specific syntax and semantics. XML and JSON are two popular data formats today [17, 33]. In the context of database management, data serialization is very relevant for several reasons: it is fundamental to support the interoperability of heterogenous databases; it allows automatic data processing; it facilitates database benchmarking as the same data can be shared among systems; it facilitates the translation to other data formats; it results in a simple backup method; other manipulation and visualization tools can read the data.

In the last years, the massive generation of large amounts of graph data has motivated the development of graph-oriented database system, most of them designed to support property graphs (i.e. labeled directed graphs where nodes and edges can have label-value properties) [11, 24, 31]. Although these systems

are very similar, they show variations in the implementation of the features presented by the property graph data model (as we will show in this paper).

The lack of a unique property graph data model directly influences the development of other components, including query languages and serialization formats. Although there are some graph data formats available (like GraphML or DotML), there is no a standard one, and none of them is able to cover all the features presented by the property graph data model.

In this paper we introduce YARS-PG, a data format to serialize property graphs. YARS-PG was designed to satisfy functional and non functional requirements. The functional requirements are related to the intrinsic features of the property graph data model. In this sense, YARS-PG is able to serialize property graphs containing multi-labeled nodes, multi-labeled edges, directed and undirected edges, mono-value and multi-value properties, and null values.

In terms of non functional requirements, we considered expressiveness, conciseness and readability. Expressiveness implies the types of objects and relationships that a serialization is able to express (i.e. data models). In this sense, YARS-PG allows to encode all the features presented by the property graph model. Conciseness is related to the number of extra syntactic elements used by the serialization. Note that such extra elements are required to parse the data in the right way. In this sense, YARS-PG provides a simple syntax with a reduced number of extra characters. Readability concerns the facilities to encode the structure of the data. In this case, YARS-PG is inspired on the syntax used by popular graph query languages (e.g. Cypher and Gremlin) to encode the structure of a property graph (i.e. nodes, edges and properties).

This article is organized as follow. First, we present a review of current graph database systems, selecting those oriented to support property graphs (Sect. 2). Next, we present a formal definition of the property graph data model, in such a way that it is general enough to cover all the data modeling features provided by a property graph (Subsect. 3.1). Such definition was used to compare current database systems (Subsect. 3.2). Next, we propose YARS-PG as a general and flexible format to serialize property graphs (Sect. 4). We present the syntax of the format and provide a comparison with other graph-oriented serialization formats (Sect. 5).

2 Review of Current Graph Database Systems

The current market of graph databases includes over 30 systems¹, most of which are designed to store and query property graphs. Table 1 shows a representative group of systems supporting property graphs. Some systems were not included for different reasons. We discard systems abandoned or no longer available, e.g. FlockDB, and GlobalsDB. We remove systems not supporting property graphs, e.g. HyperGraphDB, Graph Engine, Sqrrl, FaunaDB, and GRAKN.AI. We also discard systems focused on RDF (including Dgraph, GraphDB, Blazegraph, and Stardog). Other proposals are not strictly database systems, e.g. Giraph that is a graph processing framework, and HGraphDB that is an abstract layer.

¹ <https://db-engines.com/en/ranking/graph+dbms>.

Table 1 shows general information about the selected graph database systems. Specifically, we annotated the system’s name, license types, the programming language that the system was implemented with, supported data models, and existence of a query language. We found that the systems allow four types of licenses: GNU GPL, Apache, GNU Affero General Public License (GNU AGPL)² and Commercial. Most systems provide a commercial version, and some of them provide an open source version. The most preferred programming language for system implementation is Java, followed by C++, C, Scala and C#. Although the selected systems are based on a graph-based data model, we found that other abstractions are also supported (i.e. some systems are multi-model). Almost every system supports a query language. The most commonly used is Gremlin, followed by openCypher.

Table 1. General information about graph databases

System	License				Prog. language			Data model					Query lang.	
	GNU GPL	Apache	GNU AGPL	Commercial	C / C++	Java / Scala	C#	Graph	Relational	Object	Document	Column		Key-value
Neo4j	•			•		•		•						•
Datastax				•		•		•				•		•
OrientDB		•		•		•		•		•	•			•
ArangoDB		•		•		•		•			•			•
JanusGraph		•		•		•		•						•
Neptune				•				•						•
TigerGraph				•	•	•		•						•
InfiniteGraph				•		•		•						
InfoGrid			•			•		•						
Sparksee				•	•			•						
Memgraph				•				•						•
VelocityDB				•			•	•		•				•
AgenGraph			•	•	•			•	•					•
TinkerGraph	•					•		•						•
HGraphDB	•					•		•						•

3 The Property Graph Data Model

In the most general sense, a property graph is a directed labelled multigraph with the special characteristic that each node or edge could maintain a set of

² GNU AGPL is a free license based on the GNU GPL and it is considered for any software that will commonly be run over a network.

property-value pairs. The primary components of a property graph are nodes, edges and properties. The secondary components are labels (for nodes, edges and properties) and data types for property values.

The notion of property graph was introduced by Rodriguez and Neubauer in [26]. It is possible to find variations in the basic definition [2, 7, 12, 30], most of them related to the support for multiple labels for nodes and edges, or the occurrence of multivalued properties. In this section we provide a general definition which allows to exploit all the features of the property graph structure. Such definition is used to analyze the features covered by current graph databases systems.

3.1 Formal Definition of a Property Graph

Assume that L is an infinite set of labels (for nodes, edges and properties), and V is an infinite set of values (atomic or complex). Given a set S , we assume that $\mathcal{P}(S)$ is the power set of S , i.e. the set of all subsets of S , including the empty set \emptyset and S itself.

Definition 1. *A property graph is a tuple $G = (N, E, P, \delta, \lambda, \rho, \sigma)$ where:*

1. N is a finite set of nodes (also called vertices), E is a finite set of edges, and P is a finite set of properties, satisfying that $N \cap E \cap P = \emptyset$;
2. $\delta : E \rightarrow (N \times N)$ is a total function that associates each edge in E with a pair of nodes in N (i.e., δ is the usual incidence function in graph theory);
3. $\lambda : (N \cup E) \rightarrow \mathcal{P}(L)$ is a total function that associates a node/edge with a set of labels from L (i.e., λ is a labeling function for nodes and edges);
4. $\rho : P \rightarrow (L \times V)$ is a total function that associates each property with a pair label-value;
5. $\sigma : (N \cup E) \rightarrow \mathcal{P}(P)$ is a total function that associates each node or edge with a set of properties, satisfying that $\sigma(o_1) \cap \sigma(o_2) = \emptyset$ for each pair of distinct objects o_1, o_2 in the domain of σ .

According to the above definition: N , E and P have no elements in common; given an edge e such that $\delta(e) = (n_1, n_2)$, we will say that n_1 and n_2 are the “source node” and the “target node” of e respectively, i.e. the edges are directed; nodes and edges could have zero or more labels; each property has a single label and a single value (although it could be complex); nodes and edges could have zero or many properties, and each property belongs to a unique node or edge.

Figure 1 shows a graphical representation of a property graph. Following our formal definition, the example property graph will be described as follows:

$$\begin{aligned}
 N &= \{n_1, n_2, n_3, n_4, n_5, n_6\} \\
 E &= \{e_1, e_2, e_3, e_4, e_5, e_6\} \\
 P &= \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, \\
 &\quad p_{13}, p_{14}, p_{15}, p_{16}, p_{17}, p_{18}, p_{19}, p_{20}, p_{21}, p_{22}\} \\
 \lambda(n_1) &= \{\text{Author}\}, \sigma(n_1) = \{p_1, p_2\}, \rho(p_1) = (\text{fname}, \text{“John”}), \\
 \rho(p_2) &= (\text{lname}, \text{“Smith”}) \\
 \lambda(n_2) &= \{\text{Author}\}, \sigma(n_2) = \{p_3, p_4\}, \rho(p_3) = (\text{fname}, \text{“Alice”}),
 \end{aligned}$$

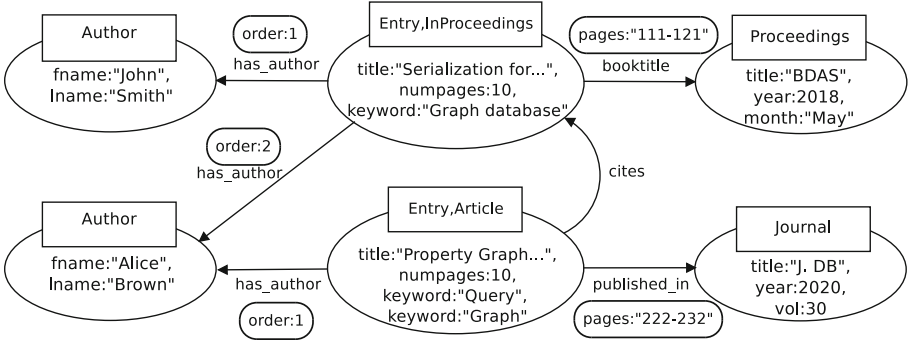


Fig. 1. Example of property graph representing bibliographic information

$$\begin{aligned}
 \rho(p_4) &= (\text{lname}, \text{"Brown"}) \\
 \lambda(n_3) &= \{\text{Entry,InProceedings}\}, \sigma(n_3) = \{p_5, p_6, p_7\}, \rho(p_6) = (\text{numpages}, 10), \\
 \rho(p_5) &= (\text{title}, \text{"Serialization for..."}), \rho(p_7) = (\text{keyword}, \text{"Graph database"}), \\
 \lambda(n_4) &= \{\text{Entry, Article}\}, \sigma(n_4) = \{p_8, p_9, p_{10}, p_{11}\}, \\
 \rho(p_8) &= (\text{title}, \text{"Property Graph..."}), \rho(p_9) = (\text{numpages}, 10), \\
 \rho(p_{10}) &= (\text{keyword}, \text{"Query"}), \rho(p_{11}) = (\text{keyword}, \text{"Graph"}), \\
 \lambda(n_5) &= \{\text{Proceedings}\}, \sigma(n_5) = \{p_{12}, p_{13}, p_{14}\}, \rho(p_{12}) = (\text{title}, \text{"BDAS"}), \\
 \rho(p_{13}) &= (\text{year}, 2018), \rho(p_{14}) = (\text{month}, \text{"May"}) \\
 \lambda(n_6) &= \{\text{Journal}\}, \sigma(n_6) = \{p_{15}, p_{16}, p_{17}\}, \rho(p_{15}) = (\text{title}, \text{"J. DB"}), \\
 \rho(p_{16}) &= (\text{year}, 2020), \rho(p_{17}) = (\text{vol}, 30) \\
 \delta(e_1) &= (n_3, n_1), \lambda(e_1) = \{\text{has_author}\}, \sigma(e_1) = \{p_{18}\}, \rho(p_{18}) = (\text{order}, 1) \\
 \delta(e_2) &= (n_3, n_2), \lambda(e_2) = \{\text{has_author}\}, \sigma(e_2) = \{p_{19}\}, \rho(p_{19}) = (\text{order}, 2) \\
 \delta(e_3) &= (n_4, n_2), \lambda(e_3) = \{\text{has_author}\}, \sigma(e_3) = \{p_{20}\}, \rho(p_{20}) = (\text{order}, 1) \\
 \delta(e_4) &= (n_4, n_3), \lambda(e_4) = \{\text{cites}\} \\
 \delta(e_5) &= (n_3, n_5), \lambda(e_5) = \{\text{booktitle}\}, \sigma(e_5) = \{p_{21}\}, \\
 \rho(p_{21}) &= (\text{pages}, \text{"111-121"}) \\
 \delta(e_6) &= (n_4, n_6), \lambda(e_6) = \{\text{published_in}\}, \sigma(e_6) = \{p_{22}\}, \\
 \rho(p_{22}) &= (\text{pages}, \text{"222-232"})
 \end{aligned}$$

3.2 Features of Current Property Graph Database Systems

Given the graph database systems presented in Sect. 2, we analyze their support for the features of the property graph data model presented above. Table 2 shows the results of our evaluation and are discussed below. We will use “some” to denote that a feature is covered by less than 50% of the systems, and “most” otherwise.

Node/Edge Labels. All the systems support labels for nodes and edges (zero, one or more). Some systems support nodes without labels, but unlabeled edges are not supported. Some systems support multiple labels for nodes, and just one system for allow many labels for edges.

Table 2. Property graph features supported by graph database systems.

System	Node labels			Edge labels			Edges				Properties			
	Zero	One	Many	Zero	One	Many	Directed	Undirected	Multiple	Duplicated	Mono-value	Multi-value	Null value	Duplicates
Neo4j	•		•	•			•		•	•		•		
Datastax		•		•			•		•	•		•		
OrientDB		•		•			•	•	•	•		•	•	
ArangoDB		•		•			•	•	•	•		•	•	
JanusGraph	•	•		•			•	•	•	•		•		
Amazon Neptune	•		•	•			•		•	•		•		
TigerGraph		•		•			•	•	•			•		
InfiniteGraph		•		•			•	•	•	•		•	•	
InfoGrid	•		•	•			•	•			•		•	
Sparksee		•		•			•	•	•	•	•		•	
Memgraph	•		•	•			•		•	•		•	•	
VelocityDB		•		•			•	•	•	•		•	•	
AgensGraph	•		•	•	•		•		•	•		•		
TinkerGraph		•		•			•		•	•		•		
HGraphDB		•		•			•		•	•	•		•	

Edges. All the systems support directed edges. More than a half of the systems allow undirected edges in an explicit way (recall that an undirected edge can be simulated with two directed edges, but the opposite is not possible). Practically all the systems allow multiple edges between a pair of nodes (i.e. they support multigraphs), and such edges could have the same label (i.e. the edges are independent of the labels).

Properties. A property is a pair $p = (l, v)$ where l is the property label (or property name) and v is the property value. A property have a single and unique label. Most systems support multivalued properties (e.g. emails for a person), a feature supported in two possible ways: properties with the same label, or properties with complex values (e.g. an array of strings). More than a half of the systems allow the *null* value to support the explicit description of an empty property³. The notion of duplicate property is not supported by current systems.

³ This feature must not be confused with the *null* values allowed in the query language provided by the system.

4 YARS-PG Serialization

In this section we describe YARS-PG, a serialization for property graphs inspired in a serialization for RDF data called YARS [29] (we compare them in Subsect. 5.1). A YARS-PG serialization contains node declarations and relationship declaration (no order is required for them).

A *node declaration* begins with the object identifier (OID) of the node, followed by a list of node labels (inside squared brackets), a colon, and the properties of the node (inside braces). A *relationship declaration* contains the OID of the source node (inside parenthesis), a set of labels, a set of properties, and the OID of the target node. Relationships can be directed (->) or undirected (-). A relationship declaration is based on paths, following the syntax used in graph query languages like PGQL [25], Cypher [22] and G-CORE [3]. YARS-PG allows cyclic relationships and multiple relationships between the same pair of nodes.

A *property* is represented as a pair $p : v$, where p is the property label and v the property value. A property value could be atomic (e.g. string, integer, float, *null*, *true*, *false*) or complex (i.e. a list of atomic values).

The following example presents YARS-PG that is also showed in Fig. 1. The example presents a graphical representation of a property graph that contains bibliographic information. The node declarations are shown in lines 1–8. The relationship declarations are shown in lines 9–15.

```

1 Author01[Author]:{fname:"John",lname:"Smith"}
2 Author02[Author]:{fname:"Alice",lname:"Brown"}
3 EI01[Entry:InProc]:{title:"Serialization for...",
4                 numpages:10,keyword:"Graph database"}
5 EA01[Entry:Article]:{title:"Property Graph...",
6                 numpages:10,keyword:["Query", "Graph"]}
7 Proc01[Proceedings]:{title:"BDAS",year:2018,month:"May"}
8 Jour01[Journal]:{title:"J. DB",year:2020,vol:30}
9
10 (EI01)-[has_author {order:1}]->(Author01)
11 (EI01)-[has_author {order:2}]->(Author02)
12 (EA01)-[has_author {order:1}]->(Author02)
13 (EA01)-[cites]->(EI01)
14 (EI01)-[booktitle {pages:"111-121"}]->(Proc01)
15 (EA01)-[published_in {pages:"222-232"}]->(Jour01)
    
```

The main railroad diagram of a node definition is presented in Fig. 2. A node declaration begins with identifier (*ido*). The next part is a node label (*node_label*) nested in square brackets. Node properties (*prop*) are located in curly brackets. A parse tree of first two nodes are presented in Fig. 3. It has interior and leaf nodes. Interior nodes (e.g. *key*, *value*) are non-terminal symbols. Leaf nodes (e.g. *Author*, *fname*) are terminal symbols.

The main railroad diagram of a relationship declaration is presented in Fig. 4. A relationship declaration begins with first identifier (*ido*) nested in round brackets. The next part is a relationship label (*relationship_label*) with relationship

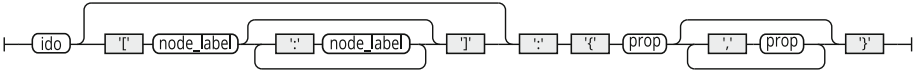


Fig. 2. Railroad diagram of a node declaration

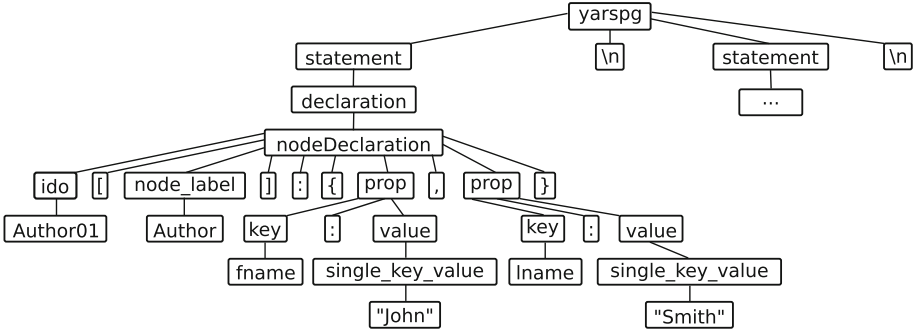


Fig. 3. Parse tree fragment of first two lines in example

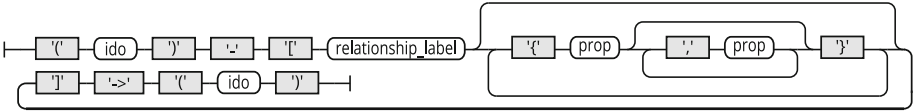


Fig. 4. Railroad diagram of directed relationship declaration

properties (`prop`). Properties begins and ends with curly brackets. The last element is the second identifier (`ido`) nested in round brackets.

The entire YARS-PG grammar in ANTLR 4 [23] and also in EBNF notation⁴ has been made available in the GitHub repository [28]. We prepare three different parsers of YARS-PG in Java [19], Python [20], and C# [18].

5 Related Work

5.1 YARS-PG Versus YARS

YARS-PG is based on YARS [29], a concise RDF serialization proposed to facilitate data exchange between RDF and property graph databases.

The following example presents a YARS serialization.

```

1  :rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  (a {value:<http://example.org/p#j>})
3  (b {value:<http://xmlns.com/foaf/0.1/Person>})
4  (a)-[:rdf:type]->(b)

```

⁴ <https://www.w3.org/TR/REC-xml/#sec-notation>.

YARS-PG and YARS are textual and path-based. There are three main differences between YARS and YARS-PG. The first difference is the abandonment of support for RDF prefixes (line 1 and line 4) because this abbreviations are not useful for property graphs. The next one is the abandonment of support for URI datatypes between the < and > characters (line 2 and line 3) because property graphs do not have a special datatype for such references. The last change concerns handling of properties in edges, which were not provided by the mapping algorithm from RDF to YARS.

5.2 Graph Serialization Formats

In the property graphs field there are a few solutions for serializing graphs. It may be divided into four groups: formats that use XML, formats that use JSON, tabular-based serializations and text-based ones. Table 3 presents supporting details of those serializations, namely: key-value pair support, multi-values support, support of *null* as a special marker, node multi-labels support, unique label support, directed edges support, undirected edges support, multi-edges with the same label support, unstructured data support, and supported types of formats i.e. XML, JSON, textual, tabular.

The first group can be distinguished to GEXF [14], GraphML [9], DotML⁵, DGML⁶ and GXL [16, 32]. Graph Exchange XML Format (GEXF) is syntax for describing complex networks structures, such as network nodes and edges, properties, hierarchies, and their associated data. It is dedicated for Gephi, which is network analysis and visualization software. Unfortunately, GEXF does not support multi-labels in nodes. Another serialization is GraphML. It supports properties for nodes and edges, hierarchical graphs, sub-graphs, and hyperedges. The advantage of this format is that it is widely adopted. However, the disadvantage, as in GEXF, is the lack of support for multi-labels and no grammar for a *null* value.

Another XML-based serialization is Dot Markup Language (DotML). This format is based on GraphViz DOT [10]. The disadvantage of this serialization is the lack of support for properties. Yet another syntax is Directed Graph Markup Language (DGML). This format supports cyclical and acyclic directed graphs. Unfortunately, DGML does not cope with most of the features considered in Table 3. The last XML-based format is Graph Exchange Language (GXL). It focuses on data interoperability between reverse engineering tools such as parsers, analyzers and visualizers. In its syntax, it is similar to GraphML and also has its disadvantages e.g. lack of support for multi-labels.

The second group are JSON-based serializations like GraphSON TinkerPop²⁷ and GraphSON TinkerPop³⁸. GraphSON is a part of TinkerPop – the open source

⁵ <http://www.martin-loetzsch.de/DOTML/>.

⁶ <https://docs.microsoft.com/en-us/visualstudio/modeling/directed-graph-markup-language-dgml-reference>.

⁷ <https://github.com/tinkerpop/blueprints/wiki/GraphSON-Reader-and-Writer-Library>.

⁸ <http://tinkerpop.apache.org/docs/current/reference/#graphson-reader-writer>.

Table 3. Comparison of property graph serializations

System	Properties			Labels		Edges				Format				
	Pairs	Multiple	<i>Null</i> value	Multiple	Unique	Directed	Undirected	Multiple	Same label	Unstructured	XML	JSON	Textual	Tabular
GEXF	•					•	•	•	•	•	•			
GDF			◦△			•		•	◦▲					•
GML	•		◦△			•	◦▽	•					•	
GraphML	•	◦			•▼	•	•	•		•	•			
Pajek NET				•		•	•	•						•
GraphViz DOT	•	◦	◦△	•		•	•		•	•			•	
UCINET DL	•			•		•	•	•	•				•	
Tulip TPL		•				•							•	
Netdraw VNA	•		◦△		•	•		•	•					•
DotML				•		•		•	•	•	•			
S-Dot				•		•		•	•	•			•	
GraphSON TP2	•					•		•	•	•		•		
GraphSON TP3	•	◦		•	•	•		•	•	•		•		
DGML	•					•		•		•	•			
GXL	•	◦			•▼	•	•	•		•	•			
YARS-PG	•	•	•	•	◦	•	•	•	•	•			•	

△ no grammar

▽ only global definition

▲ labels supported as properties

▼ only in the sense of identifiers

graph computing framework, which has its implementations for many databases. The third version of serialization, in contrast to GraphSON TinkerPop 2 supports multiple labels for nodes. However, these labels must be unique. GraphSON TinkerPop 3 also brings partial support for the possibility of defining several values for one key. Both versions support properties but do not support undirected edges. GraphSON TinkerPop 3 is not backward compatible.

There are also a few tabular-based formats including GUESS GDF [1], Pajek NET [5] and Netdraw VNA⁹. The first one is based on comma-separated values (CSV) [13] file format. GDF serialization is known from GUESS tool used to explore and visualize graphs. Blocks of declaration of vertices and edges are separated from each other. The format has rather basic capabilities and does not support properties or multiple values. Another tabular-based serialization is

⁹ <http://www.analytictech.com/Netdraw/NetdrawGuide.doc>.

Pajek NET. Serialization allows multiple labels for nodes and undirected edges, but unfortunately, does not support properties. Netdraw VNA, unlike the previously discussed serializations from this group, allows for properties and has support for multiple edges with the same label. Additionally, labels in nodes must be unique. In this serialization, similarly to NET Pajek, the values in columns are separated by spaces. Unfortunately, serialization does not allow for several values for one key, as well as multiple labels for one node.

The last group is text-based syntaxes. This group includes GML [15], GraphViz DOT [10], UCINET DL [8], Tulip TLP [4], and S-Dot¹⁰. Graph Modelling Language (GML) is a simple structure based on nested key-values lists. The purpose of the structure was to provide flexibility as a universal format. Unfortunately, GML does not support multi-values. Another serialization is Graphviz DOT. This syntax is used in various fields. The format allows to collect data, but also to stylize the graph. The disadvantage of serialization is the lack of multigraph support. Yet another syntax is UCINET DL. This format based on matrices and lists. The disadvantage of this serialization is the inability to use multi-value. The next syntax is Tulip TLP. This format has structure based on round brackets. The serialization allows to collect data, but also to stylize the graph. The last serialization belonging to textual group is S-Dot. This format is based on GraphViz DOT and on similar serialization DotML. Unfortunately, this format does not support properties.

Comparing the above serializations to YARS-PG, we can see that almost all features, listed in Table 3, are supported. The example in Sect. 4 shows key-value pairs in nodes (e.g. line 1) and in edges (e.g. line 11). This example also presents directed edges in lines 10–15, and multiple properties in line 6. Our proposal allows to use the same name of labels but the parser treats it as the same label.

6 Conclusions

This paper presents YARS-PG, a data serialization format for property graphs which is simple, extensible, and platform independent. YARS-PG supports all the features allowed by the current database systems based on the property graph data model, and can be adapted in the future to work with various database systems, visualization software and other graph-oriented tools.

The future work will focus on providing a binary and a compact version of this serialization, which will be faster and will make the serialization a good format for storing and sharing on the Web.

Acknowledgements. This work was supported by the National Science Center, Poland (NCN) under research grant Miniatura 2 for Dominik Tomaszuk. This publication has received financial support from the Polish Ministry of Science and Higher Education under subsidy granted to the University of Białystok for R&D and related tasks aimed at development of young scientists for Lukasz Szeremeta. Renzo Angles is funded by the Millennium Institute for Foundational Research on Data (Chile).

¹⁰ <http://martin-loetzsch.de/S-DOT/>.

References

1. Adar, E.: GUESS: a language and interface for graph exploration. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2006, pp. 791–800. ACM, New York (2006). <https://doi.org/10.1145/1124772.1124889>
2. Angles, R., Arenas, M., Barceló, M.A., Hogan, M.A., Reutter, M.A., Vrgoč, M.A.: Foundations of modern query languages for graph databases. *CSUR* **50**(5) (2017). <https://doi.org/10.1145/3104031>
3. Angles, R., et al.: A core for future graph query languages. In: Proceedings of the 2018 International Conference on Management of Data, SIGMOD 2018, pp. 1421–1432. ACM, New York (2018). <https://doi.org/10.1145/3183713.3190654>
4. Auber, D., et al.: TULIP 5 (2017)
5. Batagelj, V., Mrvar, A.: Pajek— analysis and visualization of large networks. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 477–478. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45848-4_54
6. Bhatti, N., Hassan, W., McClatchey, R., Martin, P., Kovacs, Z.: Object serialization and deserialization using XML. *Advances in Data Management*, vol. 1 (2000)
7. Bonifati, A., Fletcher, G., Voigt, H., Yakovets, N.: Querying graphs. In: *Synthesis Lectures on Data Management*. Morgan & Claypool Publishers (2018). <https://doi.org/10.2200/S00873ED1V01Y201808DTM051>
8. Borgatti, S.P., Everett, M.G., Freeman, L.C.: Ucinet for Windows: software for social network analysis (2002)
9. Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., Marshall, M.S.: GraphML progress report structural layer proposal. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) GD 2001. LNCS, vol. 2265, pp. 501–512. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45848-4_59
10. Ellson, J., Gansner, E.R., Koutsofios, E., North, S.C., Woodhull, G.: Graphviz and dynagraph – static and dynamic graph drawing tools. In: Jünger, M., Mutzel, P. (eds.) *Graph Drawing Software*. Mathematics and Visualization, pp. 127–148. Springer, Berlin (2004). https://doi.org/10.1007/978-3-642-18638-7_6
11. Guminska, E., Zawadzka, T.: EvOLAP graph – evolution and OLAP-aware graph data model. In: Kozielski, S., Mrozek, D., Kasprowski, P., Małysiak-Mrozek, B., Kostrzewa, D. (eds.) *BDAS 2018*. CCIS, vol. 928, pp. 75–89. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99987-6_6
12. Hartig, O.: Reconciliation of RDF* and Property Graphs. Technical reports. <http://arxiv.org/abs/1409.3288> (2014)
13. Hausenblas, M., Wilde, E., Tennison, J.: URI Fragment Identifiers for the text/csv Media Type. RFC 7111, RFC Editor, January 2014. <http://www.rfc-editor.org/rfc/rfc7111.txt>
14. Heymann, S.: Gephi. In: Alhajj, R., Rokne, J. (eds.) *Encyclopedia of Social Network Analysis and Mining*, pp. 612–625. Springer, New York (2014). https://doi.org/10.1007/978-1-4614-6170-8_299
15. Himsolt, M.: GML: a portable graph file format (1997). <http://www.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf>
16. Holt, R.C., Winter, A., Schürr, A.: GXL: toward a standard exchange format. In: Proceedings of the Seventh Working Conference on Reverse Engineering, pp. 162–171, November 2000. <https://doi.org/10.1109/WCRE.2000.891463>
17. Kangasharju, J., Tarkoma, S.: Benefits of alternate xml serialization formats in scientific computing. In: *Proceedings of the Workshop on Service-Oriented Computing Performance: Aspects, Issues, and Approaches*, pp. 23–30. ACM, New York (2007). <https://doi.org/10.1145/1272457.1272461>

18. Litman, K.: YARSPg Parser C Sharp 0.3 (GitHub), December 2018. <https://doi.org/10.5281/zenodo.2285046>
19. Litman, K.: YARSPg-Parser-Java 0.3 (GitHub), December 2018. <https://doi.org/10.5281/zenodo.2284679>
20. Litman, K.: YARSPg Parser Python 0.4 (GitHub), December 2018. <https://doi.org/10.5281/zenodo.2285247>
21. Maeda, K.: Comparative survey of object serialization techniques and the programming supports. *Int. J. Comput. Inf. Eng.* 5(12) (2011)
22. Marton, J., Szárnyas, G., Varró, D.: Formalising openCypher graph queries in relational algebra. In: Kirikova, M., Nørnvåg, K., Papadopoulos, G.A. (eds.) *ADBIS 2017*. LNCS, vol. 10509, pp. 182–196. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66917-5_13
23. Parr, T.: *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf (2013)
24. Pluciennik, E., Zgorzałek, K.: The multi-model databases – a review. In: Kozielski, S., Mrozek, D., Kasprowski, P., Malysiak-Mrozek, B., Kostrzewa, D. (eds.) *BDAS 2017*. CCIS, vol. 716, pp. 141–152. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58274-0_12
25. van Rest, O., Hong, S., Kim, J., Meng, X., Chafi, H.: PGQL: a property graph query language. In: *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems, GRADES 2016*, pp. 1–6. ACM, New York (2016). <https://doi.org/10.1145/2960414.2960421>
26. Rodriguez, M.A., Neubauer, P.: Constructions from dots and lines. *Bull. Am. Soc. Inf. Sci. Tech.* **36**(6), 35–41 (2010)
27. Sumaray, A., Makki, S.K.: A comparison of data serialization formats for optimal efficiency on a mobile platform. In: *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, pp. 1–6. ACM (2012). <https://doi.org/10.1145/2184751.2184810>
28. Szeremeta, L.: YARS-PG ANTLR4 grammar (GitHub), February 2019. <https://doi.org/10.5281/zenodo.2555898>
29. Tomaszuk, D.: RDF data in property graph model. In: Garoufallou, E., Subirats Coll, I., Stellato, A., Greenberg, J. (eds.) *MTR 2016*. CCIS, vol. 672, pp. 104–115. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49157-8_9
30. Tomaszuk, D., Pak, K.: Reducing vertices in property graphs. *PLoS ONE* **13**(2), 1–25 (2018)
31. Warchał, L.: Using Neo4j graph database in social network analysis. *Stud. Informatica* **33**(2A), 271–279 (2012). https://doi.org/10.21936/si2012_v33.n2A.147
32. Winter, A., Kullbach, B., Riediger, V.: An overview of the GXL graph exchange language. In: Diehl, S. (ed.) *Software Visualization*. LNCS, vol. 2269, pp. 324–336. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45875-1_25
33. Yusof, K., Man, M.: Efficiency of JSON for data retrieval in big data. *Ind. J. Electr. Eng. Comput. Sci.* **7**, 250–262 (2017)