PLOS ONE

RESEARCH ARTICLE

Reducing vertices in property graphs

Dominik Tomaszuk*, Karol Pąk

Institute of Informatics, University of Bialystok, Białystok, Poland

* d.tomaszuk@uwb.edu.pl

Abstract

Graph databases are constantly growing, and, at the same time, some of their data is the same or similar. Our experience with the management of the existing databases, especially the bigger ones, shows that certain vertices are particularly replicated there numerous times. Eliminating repetitive or even very similar data speeds up the access to database resources. We present a modification of this approach, where similarly we group together vertices of identical properties, but then additionally we join together groups of data that are located in distant parts of a graph. The second part of our approach is non-trivial. We show that the search for a partition of a given graph where each member of the partition has only pairwise distant vertices is NP-hard. We indicate a group of heuristics that try to solve our difficult computational problems and then we apply them to check the the effectiveness of our approach.

Citation: Tomaszuk D, Pąk K (2018) Reducing vertices in property graphs. PLoS ONE 13(2): e0191917. https://doi.org/10.1371/journal.pone.0191917

Editor: Pablo Dorta-González, Universidad de las Palmas de Gran Canaria, SPAIN

Received: May 18, 2017

Accepted: November 27, 2017

Published: February 14, 2018

Data Availability Statement: All relevant data are within the paper and Figshare: Tomaszuk, Dominik (2017): Property Graphs. figshare. https://doi.org/10.6084/m9.figshare.5325607.v1.

Funding: The authors received no specific funding for this work.

Competing interests: The authors have declared that no competing interests exist.

1 Introduction and preliminaries

Graphs are a useful and understandable form of presenting various types of data in areas such as administration, social networks, biological sciences, media, and geography. Property graphs [1] are types of graphs that enable the construction of links, relations, and attributes of particular objects. Some elements of property graphs may be similar and share the same features. These elements can be merged together, and, thanks to that, the property graph becomes smaller, simpler, and easier to process and select the data from.

In this article, we present a way of reducing nodes of a property graph. We rely on the observation that, in the property graph, there may exist elements that have completely different properties, which are non-connectable with each other, and even have disjoint neighbors. The advantage of this approach is that merging different vertex contexts significantly reduces the chance that the query will involve different vertices that have been merged. This approach is subjected to some errors resulting from merging distant vertices and quite random data because of possible relationships. These errors can be eliminated by re-querying only within the merged entities, among which there is very little dependency because vertices are distant in the original graph. Such division speeds up the execution of queries, especially those of high complexity. In our approach to finding this kind of vertex, we use graph vertex coloring methods [2, 3].

Some initial work [4–6] has been done in Resource Description Framework (RDF) [7] and Semantic Web [8], and we are trying to move these ideas to the property graph world. In the article, we also present how to find distant vertices as well as how to find and merge similar ones.

We propose an experimental method of dealing with data similarity problems in property graphs by searching for solutions known from addressing NP-complete graph problems. We also present results obtained with METIS [9] and ColPack [10] support. Our proposals contribute to enable a user who is familiar with graph databases to use and access RDF data and property graph data as well. Property graph databases often have better performance than native RDF graph stores [11–13], so it is important to enable interoperability between these two approaches.

The PG data model rests on the concept of creating directed and key/value-based graphs. It means that there is a tail and head to each edge and both vertices and edges can have properties associated with them.

Following [14, 15], we provide a formal definition below.

Definition 1 (Property Graph). *A* Property Graph *is a tuple* $PG = \langle V, E, S, P, h_e, t_e, l_v, l_e, p_v, p_e \rangle$, *where*:

1. V *is a non-empty set of vertices*,
2. E *is a multiset of edges, which are elements of* $V \times V$,
3. S *is a non-empty set of character strings*,
4. P *is the Cartesian product* $S \times S$, *where each member has a form* $p = \langle k, v \rangle$ *(property)*,
5. $h_e: E \rightarrow V$ *is a function that yields the source of each edge (head)*,
6. $t_e: E \rightarrow V$ *is a function that yields the target of each edge (tail)*,
7. $l_v: V \rightarrow S$ *is a function mapping each vertex to a label*,
8. $l_e: E \rightarrow S$ *is a function mapping each edge to a label*,
9. $p_v: V \rightarrow 2^P$ *is a function that assigns vertices to their multiple properties, and*
10. $p_e: E \rightarrow 2^P$ *is a function that assigns edges to their multiple properties*.

Note that $\langle V, E, h_e, t_e, l_e \rangle$ is an edge-labeled directed multigraph.

Example 1. *The example in* Fig 1 *presents a property graph. This graph includes the following elements*:

$$
\begin{aligned}
S &= \{name, Alice, Bob, age, 22, since, 2001, knows, alice, bob\},\\
V &= \{v_1, v_2\},\\
p_v(v_1) &= \{\langle name, Alice\rangle, \langle age, 22\rangle\},\\
p_v(v_2) &= \{\langle name, Bob\rangle\},\\
l_v(v_1) &= alice,\\
l_v(v_1) &= bob,\\
E &= \{e_1, e_2\},\\
h_e(e_1) &= bob,\\
t_e(e_1) &= alice,\\
l_e(e_1) &= knows,\\
p_e(e_1) &= \{\langle since, 2001\rangle\},\\
h_e(e_2) &= alice,\\
t_e(e_2) &= bob,\\
l_e(e_2) &= knows,\\
p_e(e_2) &= \{\langle since, 2001\rangle\}.
\end{aligned}
$$

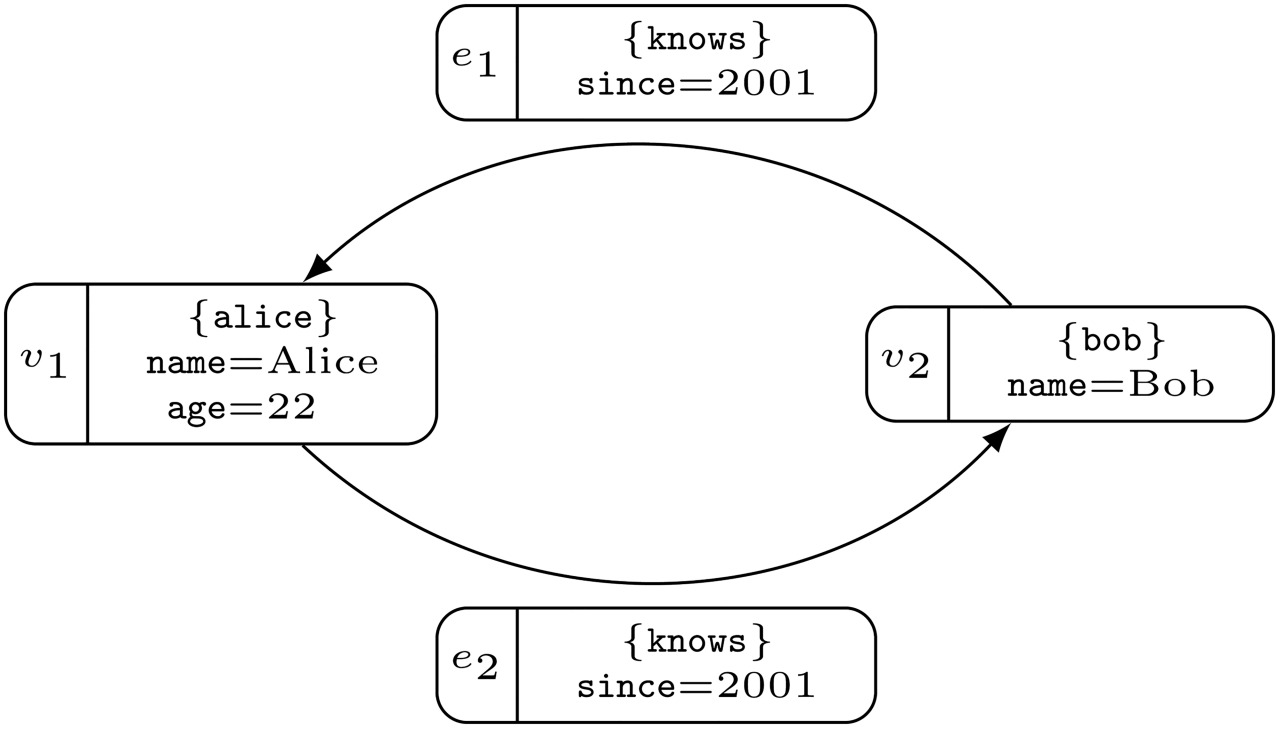

Fig 1. A property graph with two vertices, two edges and four properties. The vertex on the left has an `alice` label and a property with two key/value pairs: `name = Alice`, and `age = 22`, where `name` and `age` are keys and `Alice` and `22` are values. The edge on the top has a `knows` label and a property `since = 2001` key/value pair. The vertex on the right and the edge on the bottom are built similarly.

https://doi.org/10.1371/journal.pone.0191917.g001

Note that in Fig 1 a property ⟨name, Bob⟩ is written as `name = Bob`*, respectively. Labels (i.e. alice) are written in curly braces (i.e.* `{alice}`*). All property values start with a capital letter.*

On the other hand, RDF constitutes a universal method of the conceptual description or information modeling accessible in Web resources. The elemental constituents of the RDF data model are RDF terms that can be used in reference to resources: anything with identity. The set of RDF terms is divided into three disjoint subsets: IRIs, literals, and blank nodes. Following [7], we provide formal definitions below.

Definition 2 (IRIs). IRIs *are a set of Unicode names in registered name spaces and addresses referring to registered protocols or name spaces used to identify a resource.*

Example 2. <http://dbpedia.org/page/Dog> *is used to identify the dog in DBpedia* [16].

Definition 3 (Literals). Literals *are a set of lexical forms and datatype IRIs. A lexical form is a Unicode string, and a datatype IRI is an IRI identifying a datatype, where RDF borrows many of the datatypes defined in XML Schema 1.1* [17].

Example 3. "`1`" http://www.w3.org/2001/XMLSchema#integer, *where* `1` *is a lexical form and should be interpreted as integer number.*

Definition 4 (Blank nodes). Blank nodes *are defined as elements of an infinite set disjoint from IRIs and Literals.*

A collection of RDF triples intrinsically represents a labeled directed multigraph. The nodes are the subjects and objects of their triples. RDF is often referred to as being *graph* where each $\langle s, p, o\rangle$ triple can be interpreted as an edge $s \xrightarrow{p} o$. Several of RDF syntax (called *serializactions*) formats exist for writing down graphs. We propose Yet Another RDF Serialization (YARS), which allows prepare RDF data to exchange on the property graph data stores. One example of

such a serialization is Yet Another RDF Serialization (YARS) [15], which allow prepare RDF data to exchange on the property graph data stores.

The article is constructed according to sections. In Section 2, we motivate the need for vertex redution in property graphs. Section 3 shows that reducing vertices in property graphs is NP-hard. In Section 4, we introduce tested data sets and our experiments. Section 5 is devoted to related work. The paper ends with conclusions.

2 Motivating scenario

Let us suppose we have launched a Web crawler indexing Resource Description Framework in Attributes (RDFa) [18] data, which is a syntax that embeds *RDF triples* [7] in HTML and XML documents. Following [7], we provide definitions of RDF triples below.

Definition 5 (RDF triple). *Assume that* $\mathcal{I}$ *is the set of all Internationalized Resource Identifiers* (*IRIs*), $\mathcal{B}$ (*an infinite*) *set of blank nodes*, *and* $\mathcal{L}$ *a set of literals*. *An* RDF triple *is defined as a triple* $t = \langle s, p, o \rangle$ *where* $s \in \mathcal{I} \cup \mathcal{B}$ *is called the* subject, $p \in \mathcal{I}$ *is called the* predicate, *and* $o \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ *is called the* object.

Property graph databases are vertex-centric, whereas RDF graph stores are edge-centric. As a result, RDF graph stores use edges, many of which are not critical to our quiries, so we choose property graphs to store our data. The indexed subjects, predicates, and objects of RDF are saved in the graph database in a form of a property graph. Subjects and objects are represented as vertices, whereas predicates are edge labels. These elements can be written in YARS [15], which is a serialization for PG databases that is compatible with RDF. An exemplary property graph can be seen in Fig 2 and Example 4.

Example 4. *The example presents YARS. Lines 1–2 represent prefix directives. Lines 3–5 describe vertex declarations that is a part, where vertices are created. In lines 6–7 edges and properties are created. The example below represents two RDF triples*:

1. `:rdf`: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2. `:foaf`: <http://xmlns.com/foaf/0.1/>
3. (`a` {`value`: <http://example.org/p#j>})
4. (`b` {`value`: <http://xmlns.com/foaf/0.1/Person>})
5. (`c` {`value`: "*John Smith*"})
6. (`a`)-[:`rdf`: `type`]->(`b`)
7. (`a`)-[:`foaf`: `name`]->(`c`)

The first RDF triple consists of http://example.org/p#j (*an IRI*), http://www.w3.org/1999/02/22-rdf-syntax-ns#type (*an IRI*), *and* http://xmlns.com/foaf/0.1/Person (*an IRI*). *The second RDF triple consists of* http://example.org/p#j (*an IRI*), http://xmlns.com/foaf/0.1/name (*an IRI*), *and* `John Smith` (*a literal*). *Note that subjects are deduplicated.*

Unfortunately, both subjects and objects may be repeated in different sources that are searched by a Web crawler. For the sake of efficiency, we cannot check with every RDF triple whether nodes of the same name in the database already exist. This is why all subjects and objects of RDF triples encountered in different time intervals are entered into the database. Fig 2 shows repeated vertices and their properties. Such a state of data is not desired, because it causes difficulties with the efficiency of data processing and substantially impedes selecting the data. This is why we would like vertices that represent the same or similar thing to be represented by one node. A way of solving such problems is removing unnecessary nodes with their properties that are the same or similar and merging them into one node. Since a Web crawler

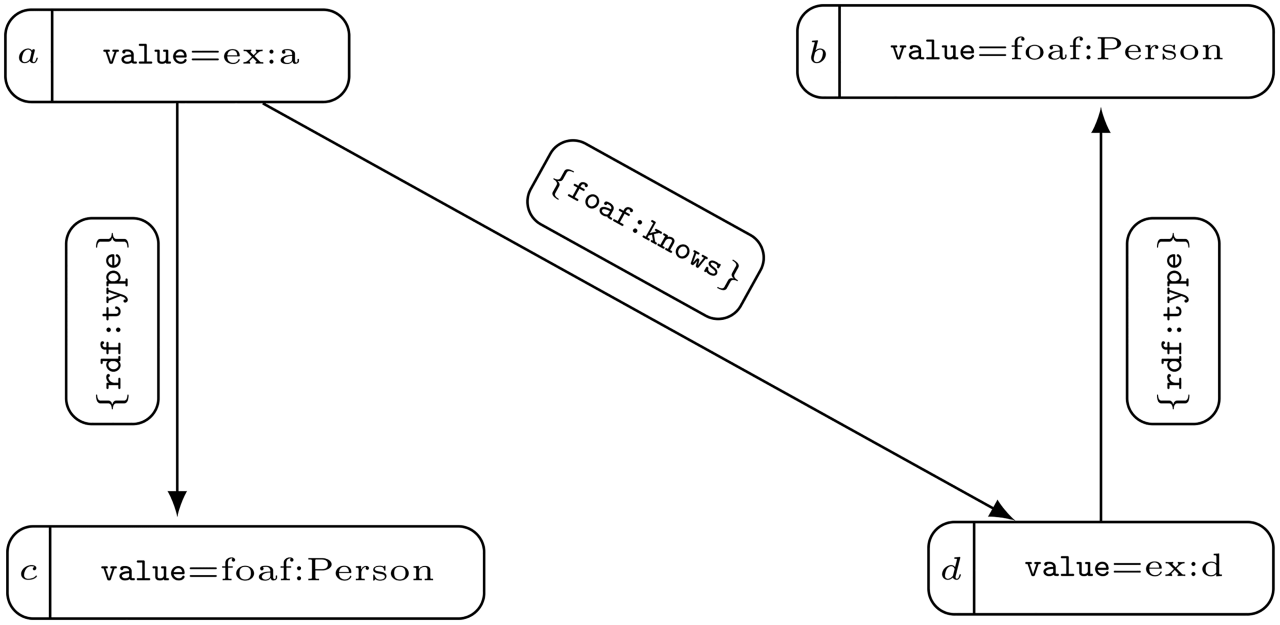

Fig 2. A property graph generated by a web crawler.

https://doi.org/10.1371/journal.pone.0191917.g002

indexes RDF triples, we assume that if the vertices have the same properties, they are not connected with the edges (see Fig 3). It is important to note that without this assumption we may encounter a loop in a modified graph, where nodes of the same property have been merged. Indeed, let us consider an edge that connects two vertices of the same property p and the vertex v_p that represents the set of all vertices with p in the modified graph. Then the edge would be

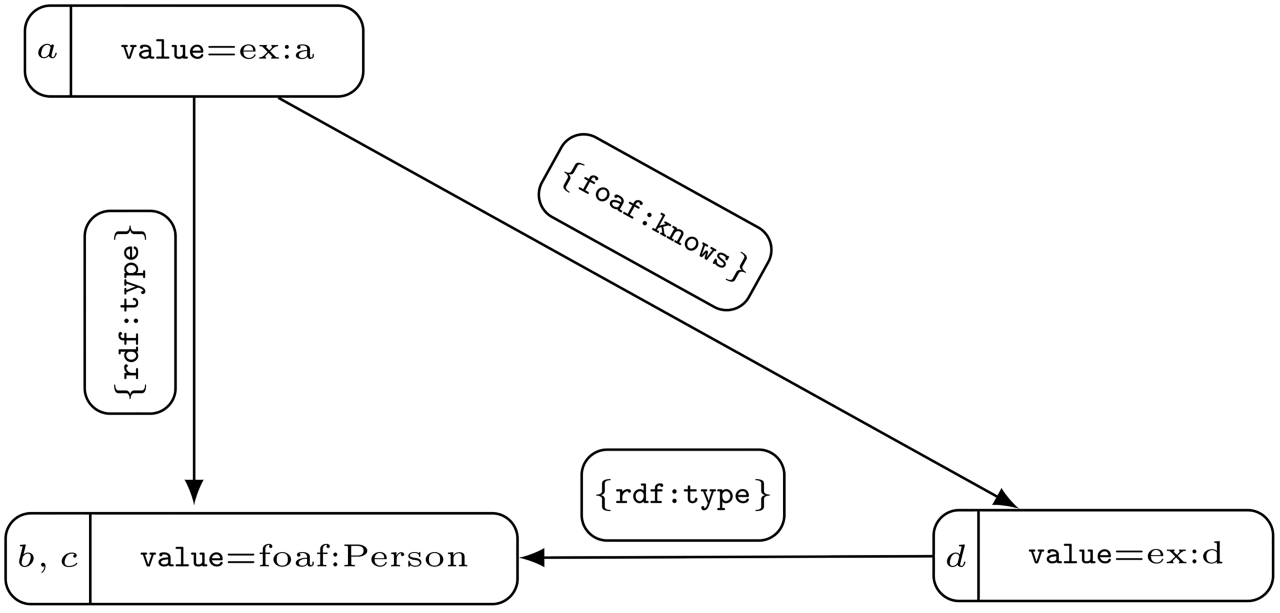

Fig 3. A modification of the graph presented at Fig 2, which is determined by a partition that retains properties as well as has the minimal number of members.

https://doi.org/10.1371/journal.pone.0191917.g003

transformed into a loop that links v_p to itself. Such loops are superfluous because of the RDF transformation algorithm to property graphs. Under assumption that there is no edge connecting two vertices with the same property, each set of vertices with a common property is independent. As a result, particular collections of merged nodes of the same property determine the partition of the graph node collection into independent sets. Because, through merging, we wanted to obtain a minimum number of nodes, we assumed that we would also allow the merging of even a few node families, where each would be defined by a common value if no two within the merged family were incidental. Elimination of vertices may take place because the data placed in the properties is similar, hence RDF subjects or objects represent the same family of RDF triple elements. Such minimization leads to a known NP-complete Graph Colorability problem (see GT4 [19]). Note that this problem is originally formulated for undirected graphs, however it is as difficult as the problem of coloring of a directed graph or even a multigraph. Moreover, subfamilies of nodes expressed by the partition of graph into independent sets do not have to be *property closed*. To say that a partition is property closed means simply that every two vertices that have the same property have to belong to the same independent sets. This property does not have to be kept even in the case of a graph of several vertices. Indeed, the graph presented in Fig 2 is 2-colorable, and such coloring is only determined by $\{\{a, b\}, \{c, d\}\}$ partition. However, this partition is not closed to the Person property (see nodes b, c). It is easily noticeable that only coloring, that retains the property, including Person, is determined by $\{\{a\}, \{b, c\}, \{d\}\}$ partition (see Fig 3).

Therefore, we can not directly use any approximation algorithm for graph coloring to reduce the number of vertices in a property graph. To solve this problem in our approach, first we transform a given property graph, sticking together vertices that have the same property as at Fig 3. Then we color a graph determined by classes of properties, and finally we assign to each vertex the color of the class to which it belongs. We describe the transformation more formally in Section 3 and use it to show in Theorem 1 that our reduction problem is NP-hard. Additionally, in Theorem 2 we show that based on this approach, we can obtain a reduced property graph that has the minimal number of vertices.

3 Reducing vertices

In this section, we will show that even in the case of narrowing the scope of searching for minimal colorings to divisions into independents sets that are closed because of a common property, the problem of minimal coloring is also an NP-hard. To do this, we will first enter the necessary definitions and notations so as to prove that our problem is NP-hard. Subsequently, we will suggest a method enabling the use of known approximating algorithms in search for minimal graph coloring. For this purpose, we will present a method of converting a property graph into an undirected graph whose minimal coloring will unequivocally designate minimal property graph coloring.

To formulate our coloring problem, we need to set the appropriate vocabulary of notations. Let $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$ be a simple undirected graph with the vertex set $\mathbb{V}$ and the edge set $\mathbb{E}$, where each edge is a two-element subset of $\mathbb{V}$. We assume that considered undirected graphs are loopless since a vertex that is incident to a loop could never be properly colored. A subset $\mathbb{V}_1$ of $\mathbb{V}$ is called independent if $\{v, u\} \notin \mathbb{E}$ for all $v, u \in \mathbb{V}_1$. For simplicity, the phrase π is a partition of $\mathbb{G}$ means that π is a partition of $\mathbb{V}$ into mutually disjoint sets.

Now we can formulate the Graph Colorability problem as the following decision problem:

Graph Colorability (GT4):

INSTANCE: An undirected graph $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$, a positive integer $k \leq |\mathbb{V}|$.

QUESTION: Does a partition of $\mathbb{G}$ into k independent sets exist?

Let $PG = \langle V, E, S, P, h_e, t_e, l_e, p_v, p_e \rangle$ be a property graph. We call *PG single*, if each vertex has exactly one property, i.e., for every $v \in V$ there exits $p \in P$ such that $p_v(v) = \{p\}$. We call *PG unique adjacency* if the adjacent vertices have no common property, i.e., $p_v(h_e(a)) \cap p_v(t_e(a)) = \emptyset$ for every $a \in E$. Note that if *PG* is single then the *unique adjacency* property can be expressed as follows: $p_v(h_e(a)) \neq p_v(t_e(a))$ for every $e \in E$. Let V_1 be a subset of V. We call V_1 *independent* if $h_e(a) \notin V_1$ or $t_e(a) \notin V_1$ for all $a \in E$. Let p be a property. We call V_1 the *class* of p if each vertex of V_1 has property p and each vertex that has property p belongs to V_1, i.e., $u \in V_1 \Leftrightarrow p \in p_v(u)$ for every $u \in V$. Finally, we call V_1 *property closed* if for every $p \in P$ whose the class has a non-empty intersection with V_1 and holds the class of p is a subset of V_1.

Combined Property Graph (**CPG**):

INSTANCE: A single property graph $PG = \langle V, E, S, P, h_e, t_e, l_v, l_e, p_v, p_e \rangle$, a positive integer $k \leq |V|$.

QUESTION: Does a partition of V into k independent property closed sets in *PG* exist?

It is important to note that there exists a partition of V into mutually disjoint independent property closed sets since the property graph *PG* is single. We show that a non-single property graph may not have such a partition. Let us consider a property graph presented at Fig 2 that contains vertices *a*, *b*, *c*. Denote by V_b an arbitrary property closed subset of vertices that contains *b*. Suppose that *b* has an additional property: `value` = ex:a (`value` is a key, and ex:a is a value of property). As *a* and *b* have the property `value` = ex:a we infer that $a \in V_b$. Similarly, $c \in V_b$ since `value` = foaf:Person is a property of *b* and *c*. But vertices *a* and *c* are connected by an edge. This contradicts our assumption that V_b is independent.

We can also consider weakened assumption that *PG* is single, preserving the existence of such a partition. Indeed, we can consider a family of property graphs where, if each pair of property closed subsets of vertices has a common member in their class, the class determined by one of them is a subset of the class for the second one. However, we can easily transform such property graphs to single ones, removing every property where the determined class is a subset of the class of another property.

Theorem 1. **CPG** *is NP-hard*.

Proof. We transform Graph Colorability to **CPG**. Let $\mathbb{G} = \langle \mathbb{V}, \mathbb{E} \rangle$ be an undirected graph and a positive integer $k \leq |\mathbb{V}|$. Without loss of generality we can assume that $\mathbb{V} = \{1, 2, \ldots, n\}$ where $n = |\mathbb{V}|$ (note that we can consider an injection function $\phi : \mathbb{V} \mapsto \{1, 2, \ldots, |\mathbb{V}|\}$ that assigns a number to each vertex of a graph).

We describe a property graph as a tuple $PG = \langle V, E, S, P, h_e, t_e, l_v, l_e, p_v, p_e \rangle$ defined by

$$\begin{aligned}
V &= \mathbb{V}, \\
E &= \{\langle v, u \rangle : \{v, u\} \in \mathbb{E} \wedge v < u\}, \\
S &= \mathbb{V} \bigcup \mathbb{E}, \\
P &= \{\langle p, q \rangle : p, q \in S\}, \\
h_e(e) &= u \text{ if } e = \langle v, u \rangle, \\
t_e(e) &= v \text{ if } e = \langle v, u \rangle, \\
l_v(v) &= v, \\
l_e(e) &= e, \\
p_v(v) &= \{\langle v, v \rangle\}, \\
p_e(e) &= \{\langle e, e \rangle\},
\end{aligned}$$

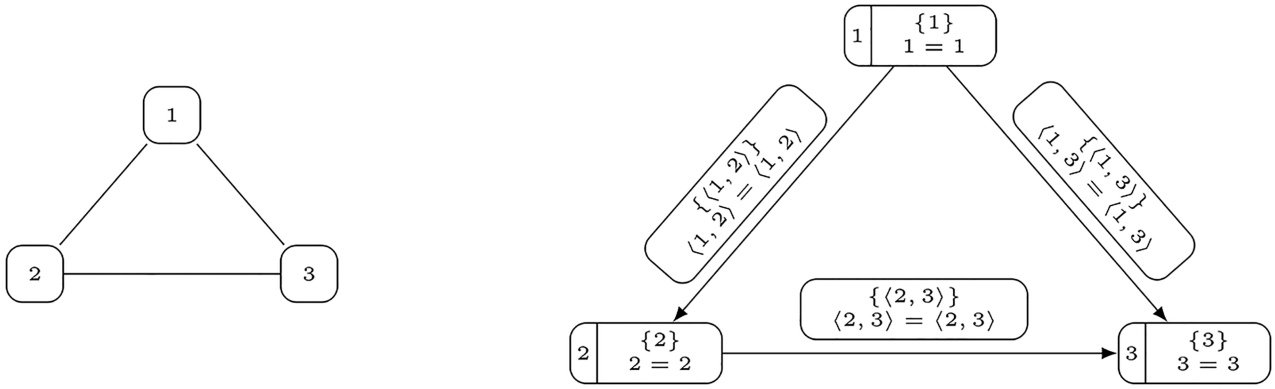

Fig 4. An illustration of the transformation construction from the proof of Theorem 1 that gives the property graph on the right hand side for an undirected graph presented on the left hand side.

https://doi.org/10.1371/journal.pone.0191917.g004

where $v, u \in V$ and $e \in E$ (see Fig 4). Note that this translation can clearly be done in LOGSPACE.

The main idea of proofs uses the fact that for every property there exists at most one vertex or one edge that has the property or more precisely

$$\forall_{p \in P} \left(|\{v \in V : p_v(v) = p\}| \leq 1 \wedge |\{e \in E : p_v(e) = p\}| \leq 1 \right). \quad (1)$$

Indeed, each property p that is assigned to a vertex or an edge has the following form $\langle x, x \rangle$, where $x \in \mathbb{V} \bigcup \mathbb{E}$. Additionally, $\mathbb{V}$ and $\mathbb{E}$ do not have common members, since each edge is a two-element subset of natural numbers and each vertex is a natural number. Based on this observation we show that there exists a partition of $\mathbb{G}$ into k independent sets if and only if there exists a partition of PG into k independent property closed sets.

Let us consider a partition π of $\mathbb{G}$ into k independent sets. Consider $\pi' = \pi$ as a partition of PG since $(V = \mathbb{V})$. We show that π' is a partition of PG into k independent property closed sets. To do this, first, we show that each member of π' is an independent set of PG. Let us consider an arrow $a \in E$ and suppose contrary that exists $V_1 \in \pi'$ such that $h_e(a) \in V_1 \wedge t_e(a) \in V_1$. The arrow a can be represented in the form $\langle v, u \rangle$ for some $v, u \in V$ where $\{v, u\} \in \mathbb{E}$ and $v < u$. Then $h_e(a) = u, t_e(a) = v$, so imply that $\{v, u\}$ is an edge that connects two members of V_1, but V_1 is an independent set of $\mathbb{G}$ as a member of π, contradiction. We are now in a position to show that each member of π' is a property closed set. But the justification of the property is obvious, since there is no two vertices that have the same property by (1), and the proof of the first implication is complete.

Let us consider a partition π_{PG} of PG into k independent property closed sets. Similarly, let us consider $\pi'_{PG} = \pi_{PG}$ as a partition of $\mathbb{G}$. Suppose, contrary to our claim that π'_{PG} is not a partition of $\mathbb{G}$ into independent sets. Then there exists $V_2 \in \pi'_{PG}$ and two vertices $v, u \in V_2$ such that $\{v, u\} \in \mathbb{E}$. Without loss of generality we can assume that $v < u$. Then $\langle v, u \rangle \in E$ connects two members of V_2 being an independent set of PG, since $h_e(\langle v, u \rangle) = u, t_e(\langle v, u \rangle) = v$, and the proof is complete.

Let $PG = \langle V, E, S, P, h_e, t_e, l_v, l_e, p_v, p_e \rangle$ be a property graph. Let π_{PG} be a set of non-empty subsets of V such that V' is a member of π_{PG} if and only if there exists a property $p \in P$ for which V' is the set of all vertices of V that have property p. Obviously, if PG is single, then π_{PG} is a partition of PG. Moreover, if PG is unique adjacency then, by definition, there are no two vertices that are connected by an arrow and belong to the same member of π_{PG}. Hence, each member of π_{PG} is an independent set for every unique adjacency property graph PG. For

subsets V_1, V_2 of V we use the following notation $V_1 \sim_{PG} V_2$ if there exists an edge $e \in E$ that connects a member of V_1 with a member of V_2 or more precisely

$$V_1 \sim_{PG} V_2 \Leftrightarrow \exists_{e \in E}(h_e(e) \in V_1 \wedge t_e(e) \in V_2) \vee (h_e(e) \in V_2 \wedge t_e(e) \in V_1). \quad (2)$$

Now, we define a partition graph $\mathcal{G}(PG, \pi)$ for an arbitrary partition π of V as follows. The set of vertices in $\mathcal{G}(PG, \pi)$ equals π and two vertices V_1, V_2 of $\mathcal{G}(PG, \pi)$ are connected if $V_1 \neq V_2$ and $V_1 \sim_{PG} V_2$.

Let us consider a partition σ of the set of vertices in $\mathcal{G}(PG, \pi)$ and a member $V' \in \sigma$. Note that $\bigcup_{x \in V'} x$ is a subset of V, since each member of σ is a set of nonempty subsets of V. In consequence,

$$\mathcal{P}^V(\sigma) = \{\bigcup_{x \in V'} x : V' \in \sigma\} \quad (3)$$

is a set of nonempty subsets of V. Note that $\mathcal{P}^V(\sigma)$ is also a partition of $\mathbb{G}$. Indeed, $\mathcal{P}^V(\sigma)$ covers the whole V since, for each vertex $v \in V$ there exist $V' \in \pi$, $V'' \in \sigma$ such that $v \in V' \in V''$, hence $v \in \bigcup_{x \in V''} x$. Additionally, if two members of $\mathcal{P}^V(\sigma)$, e.g., $\bigcup_{x \in V_1''} x$, $\bigcup_{x \in V_2''} x$, have a common vertex v, then there exist $V_1' \in V_1''$, $V_2' \in V_2''$ such that $V_1' \ni v \in V_2'$. But π, σ have mutually disjoint members, hence $V_1' = V_2'$ and $V_1'' = V_2''$, and finally these two members of $\mathcal{P}^V(\sigma)$ are equals.

Similarly, if π is property closed then each member V' of π that has a common element with a member V'' of π_{PG}, has to contain whole V'', i.e., $V'' \subseteq V'$. In consequence, we can assign a partition of $\mathcal{G}(PG, \pi_{PG})$ to the partition π. This partition is defined as follows:

$$a \in \mathcal{P}^{\pi_{PG}}(\pi) \Leftrightarrow \exists_{V' \in \pi} a = \{V'' \in \pi_{PG} : V'' \subseteq V'\}. \quad (4)$$

We show that based on $\mathcal{G}(PG, \pi_{PG})$ we can adapt each known approximation algorithm for graph coloring to **CPG** problem.

Theorem 2. *Let PG be single unique adjacency property graph. Then*

(i) *for every π be a partition of PG into k independent property closed sets holds $\mathcal{P}^{\pi_{PG}}(\pi)$ is a partition of $\mathcal{G}(PG, \pi_{PG})$ into k independent sets, and*

(ii) *for every π be a partition of $\mathcal{G}(PG, \pi_{PG})$ into k independent sets holds $\mathcal{P}^V(\pi)$ is a partition of PG into k-independent property closed sets.*

Proof.

(i) Let π be a partition of PG into k independent property closed sets. Obviously, $\mathcal{P}^{\pi_{PG}}(\pi)$ is a partition of $\mathcal{G}(PG, \pi_{PG})$ and has k members, which is clear from (4). We showed that each member of $\mathcal{P}^{\pi_{PG}}(\pi)$ is an independent set. Let us consider $a \in \mathcal{P}^{\pi_{PG}}(\pi)$ and suppose contrary to our claim, that there exist $x, y \in a$ for which $\{x, y\}$ is an edge of $\mathcal{G}(PG, \pi_{PG})$. From (2), there exists $e \in E$ such that $h_e(e) \in x \wedge t_e(e) \in y$ or $h_e(e) \in y \wedge t_e(e) \in x$. Without loss of generality we can assume that $h_e(e) \in x \wedge t_e(e) \in y$. Additionally, from (4), there exists $V' \in \pi$ satisfying $a = \{V' \in \pi_{PG}: V'' \subseteq V'\}$. Hence $x, y \subseteq V'$, and finally $h_e(e), t_e(e) \in V'$. This contradicts the fact that V' as a member of π is independent.

(ii) Let π be a partition of $\mathcal{G}(PG, \pi_{PG})$ into k independent sets. It is evident that $\mathcal{P}^V(\pi)$ is a partition of PG and has k members. Let us consider a member a of $\mathcal{P}^V(\pi)$. Then there exist $V' \in \pi$ satisfying $a = \bigcup_{x \in V'} x$.
We show first that a is an independent property closed. Suppose first that a is not independent. Then there exists an edge $e \in E$ such that $h_e(e) \in a \wedge t_e(e) \in a$, and in consequence

there exist $x_1, x_2 \in V'$ with the following properties: $h_e(e) \in x_1$ and $t_e(e) \in x_2$. But then from (2), we obtain that $\{x_1, x_2\}$ is an edge of $\mathbb{G}(PG, \pi)$, and in consequence V' contains two connected element x_1, x_2. This contradicts the fact that V' as a member of π is an independent set.

Now we prove that a is property closed. Let us consider $q \in P$ whose class has a non-empty intersection with a. To be specific there exists $u \in a$ with $qinp_v(u)$. Then there exists $x_1 \in V'$ such that $u \in x_1$. Moreover, $x_1 \subseteq (\bigcup_{x \in V'} x) = a$, and x_1 as a member of V' is an element of π_{PG}, hence x_1 is a class of a property $q_1 \in P$ and $q_1 \in p_v(u)$. But since PG is single, u has exactly one property, therefore $q = q_1$ and in consequence the class of q, which is equal to x_1, is a subset of a. This concludes the proof.

4 Experiments and evaluation

In this section, we will present a description of data sets and experiments from three areas. The first group of experiments (Subsection 4.2) shows property graph coloring characteristics. In Subsection 4.3, we show how our proposal works on graph databases. The last group of experiments (Subsection 4.4) presents how property graph serializations deal with our solutions.

4.1 Data sets description and set-up

All experiments were executed on an Intel Core i7-4770K CPU @ 3.50GHz (4 cores, 8 threads), 8GB of RAM (clock speed: 1600 MHz), and an HDD with reading speed rated at ∼160 MB/sec (we test it in `hdparm -t`). We used Linux Mint 17.3 Rosa (kernel version 3.13.0), Python 3.4.3 with RDFLib 4.2.1, gcc 4.8.4, and Docker 1.12.3.

We gathered data sets from the Web in five ways: crawled data via modified LDSpider [20], subset of DBpedia [21], subset of Wikidata [22], W3C Public Mailing List Archives (https://lists.w3.org/) and automatically generated using Berlin SPARQL Benchmark (BSBM) [23].

The first data set (DS_1) was generated in LDSpider, which was extended with YARS support. The data set mainly concerns Friend of a Friend (FOAF) [24] information because we used FOAF URIs in the seed file. FOAF is a vocabulary that describes persons, their activities and relations with other people and objects. The next data set (DS_2) was created on the basis of DBpedia 3.0 [21], which contains data from different infoboxes in Polish. The third data set (DS_3) is a dump (access date: 2016-06-21) at the class hierarchy of Wikidata Properties [22], which is structured data of its Wikimedia sister projects including Wikipedia, Wikisource, and others. The fourth data set (DS_4) was found on the W3C Public Mailing List (https://goo.gl/9x2WAu). This data set was generated in cwm.py (that is a data processor and reasoner for the Semantic Web and has data about class/property equivalences and other Web Ontology Language [25] metadata. The last data set (DS_5) is generated using Berlin SPARQL Benchmark (BSBM) [23], which was extended with YARS support. Description of data sets is presented in Table 1.

DS_2, DS_3 and DS_4 are serialized RDF, so we had to transform thom into YARS, which is indirect PG serializaction for RDF data. Our transformation tool was built in Python, and it is available at https://github.com/domel/yars. All of the considered data sets use YARS so that they can be compared under the same conditions. Description of data sets is presented in Table 1. The first part of the table shows input files characteristics such as N-Triples size (S^b_{yars}), YARS size before removing the repeated nodes (S^b_{yars}), YARS size after removing the repeated nodes (S^a_{yars}). Generation times are presented in the second part of the table. The third part presents reducing duplicates ratio (R_d) of S^a_{yars} to S^b_{yars}. The last part of the table shows graph characteristics.

Table 1. Description of data sets.

Characteristics	DS_1	DS_2	DS_3	DS_4	DS_5
RDF triples	n/a	27063	2691	5506	n/a
File sizes					
S_{nt} [B]	n/a	3780046	362172	713761	n/a
S^b_{yars} [B]	3776452	8180057	816951	1523455	9084098
S^a_{yars} [B]	2386717	4865661	564074	786257	4940874
RDF →YARS [s]	n/a	4.318	0.370	0.629	n/a
R_d [%]	63.20	59.48	69.05	51.61	54.39
Property graph entities					
Vertices	23690	54124	5382	10228	45252
Unique vertices	9611	16777	2519	1142	8623
Edges	10876	26649	2691	3970	21104

https://doi.org/10.1371/journal.pone.0191917.t001

4.2 Property graph coloring

Our tool for property graph coloring uses ColPack [10] and METIS [9] file format. Our tool for METIS and other graph formats was built in C++ and it is available at https://github.com/domel/graph_syntax. Our main tool for PG coloring is available at https://github.com/domel/pg_color.

To reduce vertices in property graphs we have to take five steps:

1. remove deduplicate vertices,
2. transform YARS into METIS,
3. color graph,
4. reduce vertices,
5. transform METIS into YARS.

In Table 2, we present transformation from YARS into METIS and its characteristics. The first part of the table shows input files characteristics such as YARS size before removing the repeated nodes (S^b_{yars}), YARS size after removing the repeated nodes (S^a_{yars}), and METIS size (S_{metis}). In the second part of the table, we present the arithmetic mean time of transformation from 10 runs. The results show that the regularity of the graph has a strong influence on the transformation time, i.e., DS_2 is irregular in its structure and its transformation time is worse than DS_5, which is benchmark generated. The third part of the table shows output file characteristics such as YARS with color metadata size (${}_oS^{cm}_{yars}$), regular YARS file size (${}_o\,S_{yars}$), and Lzip/DEFLATE [26], tANS (ZStandard implementation: http://facebook.github.io/zstd/) [27] and Brotli [28] stream compressed YARS (${}_oS^{lzip}_{yars}$, ${}_oS^{tans}_{yars}$ and ${}_oS^{br}_{yars}$). The last part shows reducing ratios referring to YARS with color metadata (R^c) and YARS after removing the repeated nodes (R^a). In addition, we show the ratio of YARS before removing the repeated nodes (R^b) and YARS after removing the repeated nodes with compression (R^a_{lzip}, R^a_{tans} and R^a_{br}). Database servers [29–31] often can send messages to clients via HTTP [32] or RESTful web services [33]. Therefore, we decided to test our solution for compression to improve transfer speed and bandwidth utilization.

To test our approach we have to choose the minimum distance that has to be maintained between each pair of vertices merged together. At the same time, we have to choose the

Table 2. Summary of experiments.

Characteristics	DS_1	DS_2	DS_3	DS_4	DS_5
File sizes					
S^b_{yars} [B]	3776452	8180057	816951	1523455	9084098
S^a_{yars} [B]	2386717	4865661	564074	786257	4940874
S_{metis} [B]	95137	245120	18005	29108	180054
Times					
YARS → METIS [s]	3.548	13.188	0.272	0.106	3.574
Coloring time [s]	0.092	2.906	0.365	0.073	0.612
Reducing time [s]	2.728	9.611	0.364	0.129	2.915
Total time [s]	6.368	25.705	1.001	0.308	7.101
File sizes					
${}_oS^{cm}_{yars}$ [B]	2793367	5734691	669874	906578	5555733
${}_o S_{yars}$ [B]	1390302	2826548	361010	452664	3439329
${}_oS^{lzip}_{yars}$ [B]	170443	375414	24014	48676	709976
${}_oS^{tans}_{yars}$ [B]	141914	312572	19971	40552	591215
${}_oS^{br}_{yars}$ [B]	116613	261997	14403	36862	537261
Reducing ratios					
R^c [%]	73.97	70.11	82.00	59.51	61.16
R^a [%]	58.25	58.09	64.00	57.57	69.61
R^b [%]	36.82	34.55	44.19	29.71	37.86
R^a_{lzip} [%]	4.51	4.59	2.94	3.20	7.82
R^a_{tans} [%]	3.76	3.82	2.44	2.66	6.51
R^a_{br} [%]	3.09	3.20	1.76	2.42	5.91

https://doi.org/10.1371/journal.pone.0191917.t002

minimum distance between vertices of the same color. Below we present two cases: distance-1 and distance-2 [2].

In Table 3, we present distance-1 coloring characteristics, such as unique vertex reducing ratio (R^u) of unique vertices to output vertices, total vertex reducing ratio (R^t) of input vertices to output vertices. Figs 5 and 6 present DS_1 and DS_4 before and after our reducing vertices. To visualize these graphs we use the Yifan Hu algorithm (before) [34] and the Fruchterman-Reingold algorithm (after) [35].

In Table 4, we present distance-2 coloring characteristics, such as unique vertex reducing ratio (R^u), total vertex reducing ratio (R^t), which show, how many nodes of the same color are removed. Figs 7 and 8 present DS_2 and DS_3 before and after us reducing vertices. To visualize these graphs we use the Yifan Hu algorithm.

4.3 Querying graph databases

In this subsection, we show our experiments considering querying DS_1, DS_2, DS_3, DS_4 and DS_5 before and after us reducing vertices in common graph databases, such as Neo4j [30], Titan 1.0.0 [29], and OrientDB 2.2.14 [31]. All these property graph databases were running in Docker [36], which is a completely sandboxed virtual environment. The instructions to build images (i.e. *dockerfiles* are avaliable on:

1. Neo4j 3.0.1: https://hub.docker.com/_/neo4j/,
2. Titan 1.0.0: https://hub.docker.com/r/elubow/titan-gremlin/,
3. OrientDB 2.2.14 https://hub.docker.com/_/orientdb/.

Table 3. Distance-1 coloring.

Characteristics	DS_1	DS_2	DS_3	DS_4	DS_5
Vertices (input)	23690	54124	5382	10228	45252
Unique vertices (input)	9611	16777	2519	1142	8623
Vertices (output)	15	5	4	24	4
Vertex reducing ratios					
R^u [%]	0.16	0.03	0.16	2.10	0.05
R^t [%]	0.06	0.01	0.07	0.23	0.01

https://doi.org/10.1371/journal.pone.0191917.t003

We focus mainly on the speed-up of querying, i.e., the ratio of time before to after our reducing for every query: Q_1, Q_2, . . ., Q_8. In Figs 9, 10, 11 and 12, we present queries in three different databases. OrientDB has been tested using Gremlin [37] and SQL via a console. In Neo4j, we used Cypher Query Language [38], and, in Titan, we executed Gremlin queries. As we expected to reduce the number of vertices, we generally sped-up the process of querying. Indeed, the average amount of speed-up is 24.99 times. However, the speed-up is significantly dependent on the choice of a graph database. This parameter is 2.73 in Neo4j, 38.78 in Titan, 48.52 in OrientDB via Gremlin, and 9.92 in OrientDB via a console. Based on OrientDB via Gremlin we obtain the biggest average number of speed-up, however the ratio of time is greater that 1 only in 35% of querying. Note for comparison that Neo4j in 75%, OrientDB via a console in 78.7%, Titan in 82.5%, obtains speed-up greater that 1. If we compare the ratio of time, we can also compare the ratio of time in different graph databases for a particular query. Titan in 52.5%, OrientDB via a console in 25%, Neo4j in 25%, OrientDB via Gremlin in 7.5% of questions obtains the maximum speed-up based on our modification of property graphs. Additionally, OrientDB via Gremlin in 45%, OrientDB via console in 22.5%, Neo4j in 22.5%, Titan in 10% obtain the minimum speed-up.

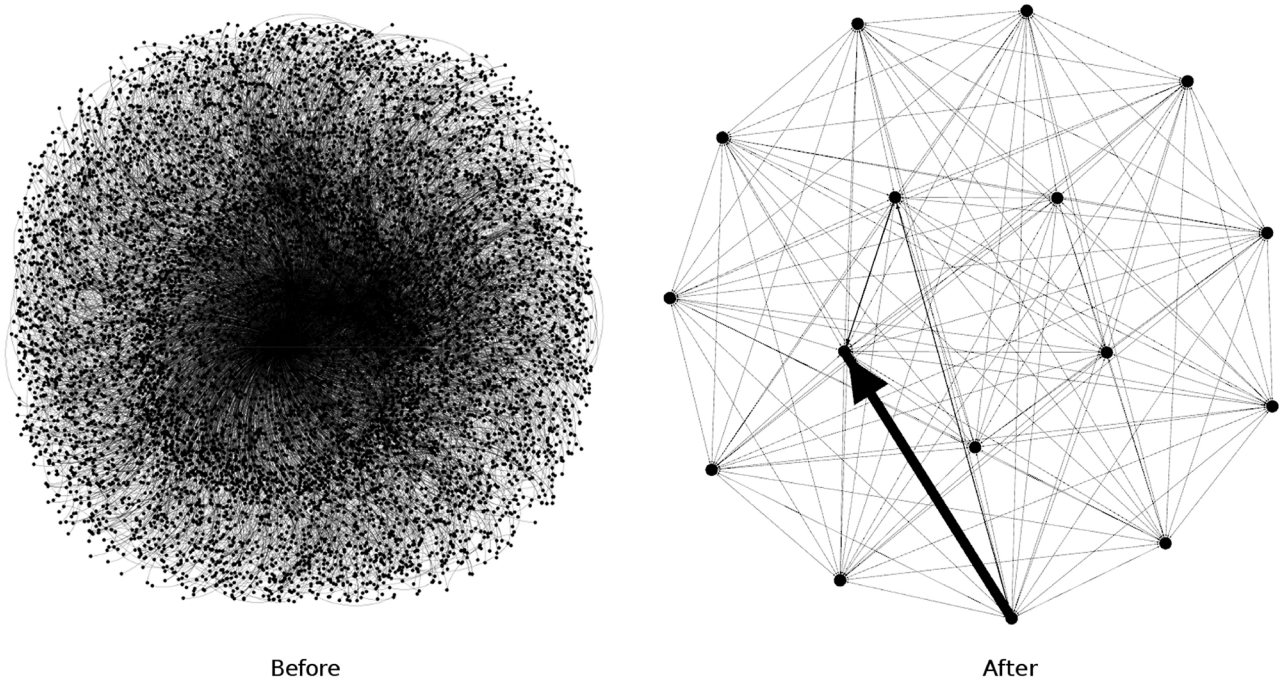

Fig 5. Distance-1 coloring of DS_1.

https://doi.org/10.1371/journal.pone.0191917.g005

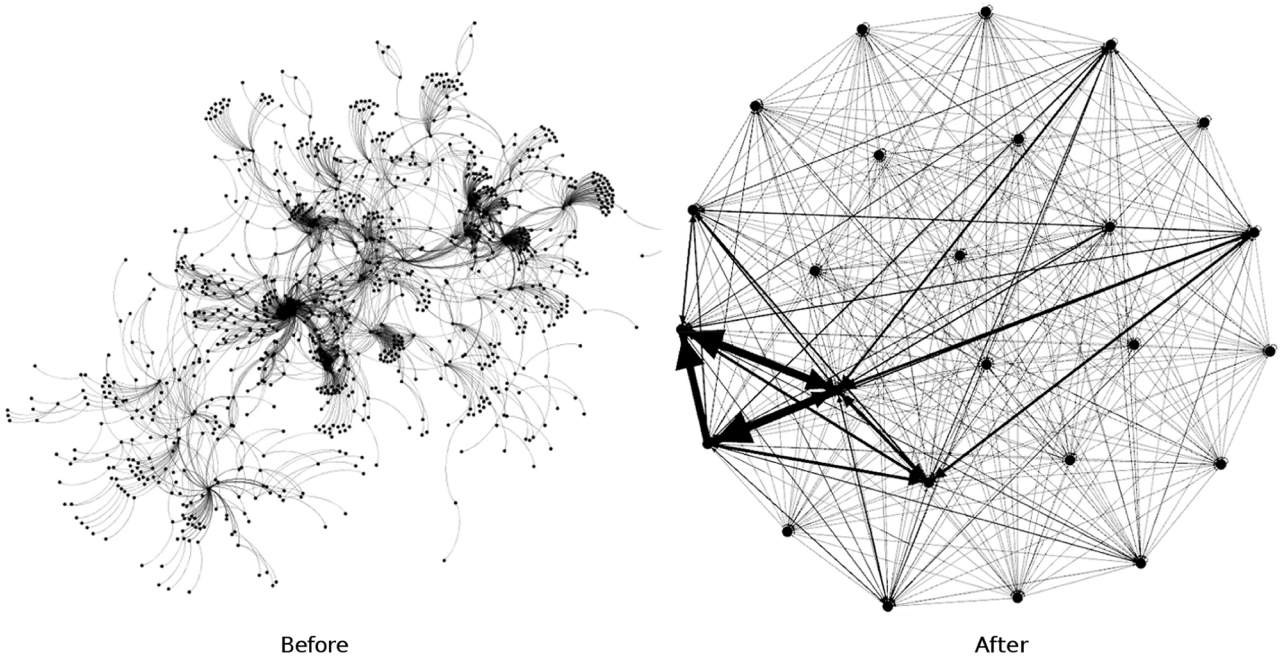

Fig 6. Distance-1 coloring of DS_4.

https://doi.org/10.1371/journal.pone.0191917.g006

Our queries can be divided into two types. The first one checks basic operations (loading data—Q_1, select all nodes with properties—Q_2, select all the edges with all properties—Q_3, removing data—Q_4). The second one checks finding the shortest path (Q_5—max depth 2, . . .Q_8—max depth 5). The analysis showed that our reduction of the number of vertices has different effects in each individual graph database that we considered. Indeed, the average speed-up of querying obtain 96.51 times in the case of the first type of queries in OrientDB via Gremlin, even if the average speed-up in the case of the second type of queries in the graph database is 0.53, i.e., the shortest paths are found about two-times slower after the modification than prior to it. However, such situations occur only in the case of OrientDB via Gremlin. The best results are obtained in the case of Titan, where the average speed-up is 30.90 times in the case of the first type of queries and 46.67 in the second one.

4.4 Serializations

In this subsection, we compare efficiency of our reducing vertices in common graph serialization formats, such as GML [39], GraphML [40], and GEXF (https://gephi.org/gexf/format/). This is important because format affects the effective property graph data storage and

Table 4. Distance-2 coloring.

Characteristics	DS_1	DS_2	DS_3	DS_4	DS_5
Vertices (input)	23690	54124	5382	10228	45252
Unique vertices (input)	9611	16777	2519	1142	8623
Vertices (output)	2036	2680	2486	127	1285
Vertex reducing ratios					
R^u [%]	21.18	15.97	98.69	11.12	14.9
R^t [%]	8.54	4.95	46.19	1.24	2.84

https://doi.org/10.1371/journal.pone.0191917.t004

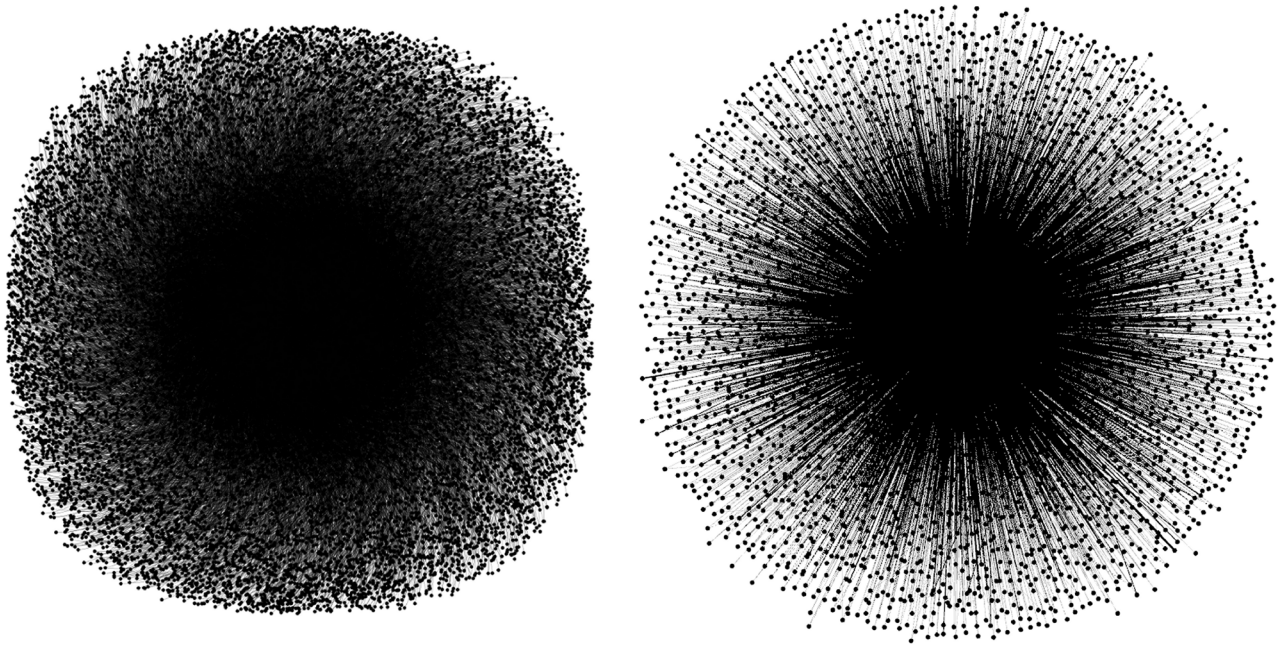

Fig 7. Distance-2 coloring of DS_2.

https://doi.org/10.1371/journal.pone.0191917.g007

transmission. This feature is useful for saving disk space, better use of database memory buffers or faster communication between a database client and a server.

Table 5 shows distance-1 characteristics of serializations before and after our vertices reduction. All results are better after our transformation. The best ratio belongs to GML (DS_1, DS_2, DS_4 and DS_5) and GEXF (DS_3).

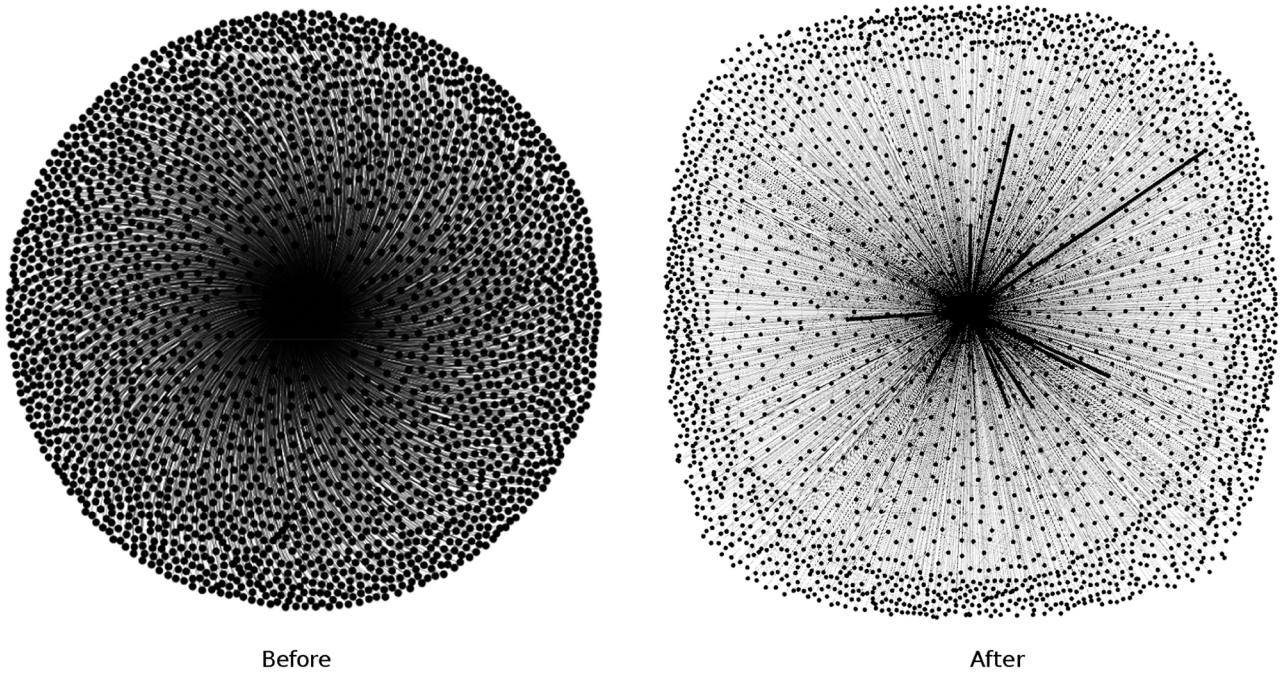

Fig 8. Distance-2 coloring of DS_3.

https://doi.org/10.1371/journal.pone.0191917.g008

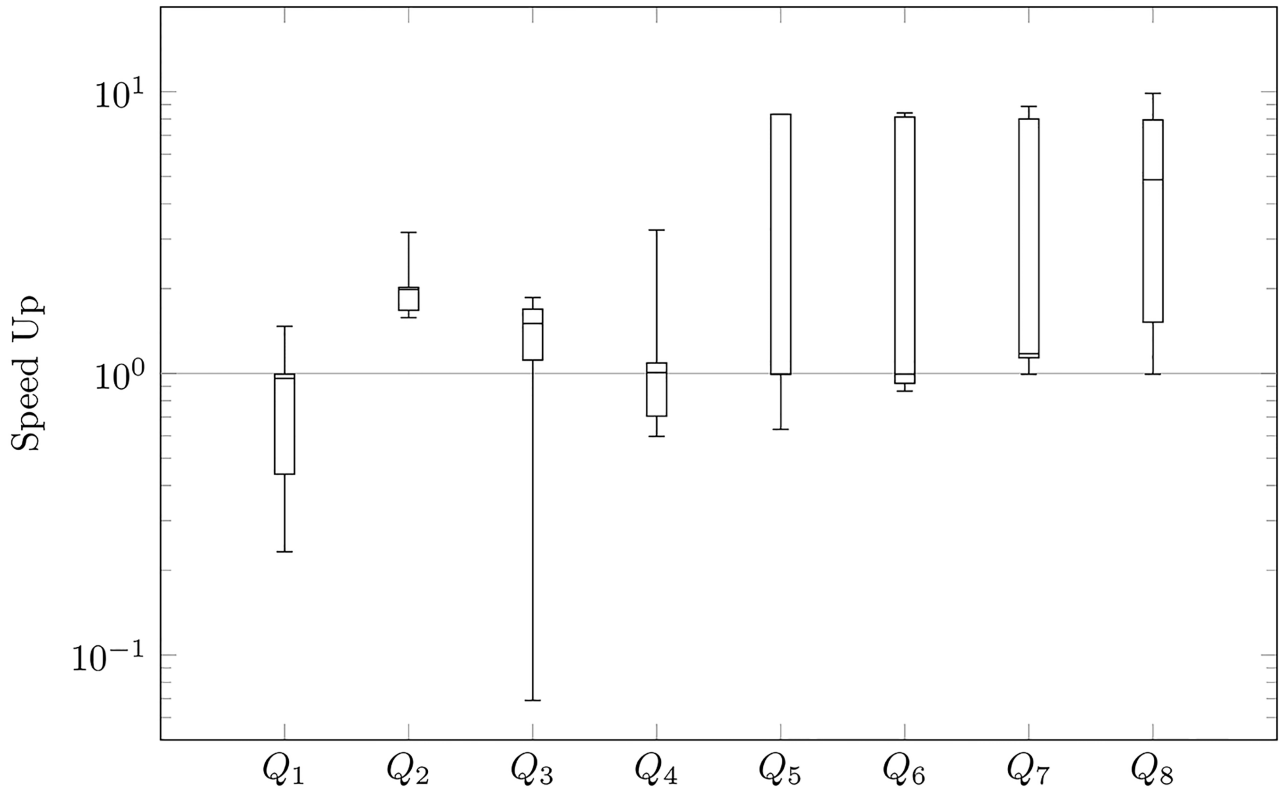

Fig 9. Speed-up of Neo4j querying.

https://doi.org/10.1371/journal.pone.0191917.g009

In Table 6, we introduce distance-2 characteristics of serializations before and after our vertices reduction. Only two cases have slightly worse results (GML in DS_1 and DS_3). It is evident that additional information about the original graph (especially information about the edges) has a great impact in case of a small reduction of nodes. The rest of the serializations give slightly better results than before our transformations.

5 Related work

5.1 Property graphs

In this section, we discuss property graph approaches and solutions. They can be divided into four groups: abstraction layer and formalization of property graphs, property graphs databases, multi-modal databases that support property graphs, and distributed processing frameworks.

The first group relates to proposals that formalize property graphs [29, 41–43]. In [41], Hartig proposes a formalization of the PG model and introduces transformations between PGs and RDF* [44]. Unfortunately, this model is not widely supported by graph stores. In [29], Jouili *et al*. suggested another definition of PG based on Blueprints (https://github.com/tinkerpop/blueprints/wiki). The PG definition is restrictive, because it assumes that labels must be unique. In this paper, authors present a distributed graph database comparison framework. In [42], Schätzle *et al*. present a formalization of PG in the RDF context. Moreover, the paper introduces a SPARQL query processor for Hadoop called S2X. Unfortunately, this paper focuses on distributed storage and do not formalize property graphs in the graph database context. In [43], Batarfi *et al*. propose a formalization of attributedgraphs, which is similar to PGs.

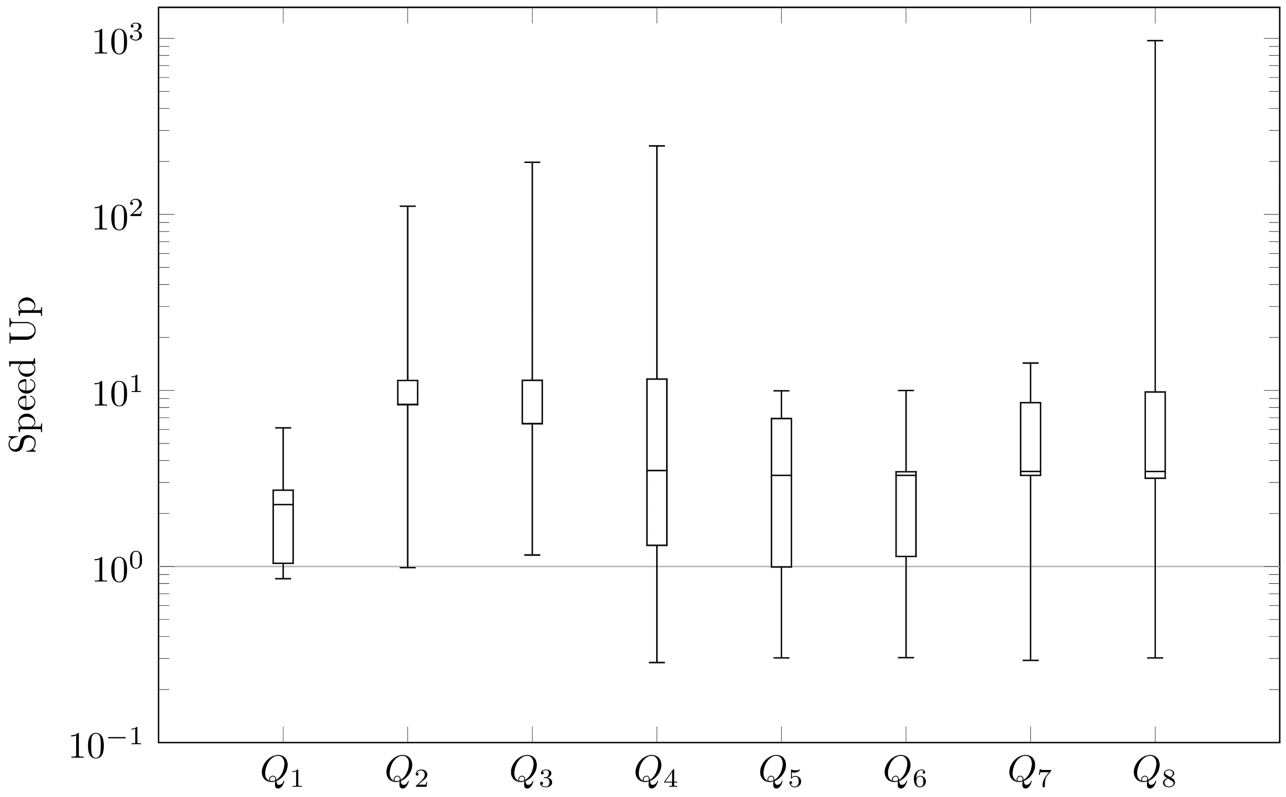

Fig 10. Speed-up of Titan querying.

https://doi.org/10.1371/journal.pone.0191917.g010

There are a few data stores in the Property Graph world [30, 45]. Neo4j [30] is a native graph database purpose-built to leverage not only data but also its relationships. Titan [29] is another graph database that is distributed and transactional. Dex/Sparksee [45] is yet another graph database, which supports data constraints to guarantee the integrity of data and relationships among them.

The third group is a multi-modal database that supports property graphs, and it can be divided into two subgroups: graph databases that support RDF and PG models [46, 47], and databases that support documents and graphs [31, 48]. Oracle database with Oracle Spatial and Graph option [46] is an add-on database feature with advanced spatial capabilities enabling the development of complex geographic information systems. Bigdata/Blazegraph [47] is another graph database that supports RDF and PG models. It is an ultra-scalable and high-performance database that supports up to $50 \cdot 10^9$ edges on a single machine. The next subgroup is OrientDB [31] and AranoDB [48]. The first solution supports different indexes and ACID transactions guaranteeing that all database transactions are processed reliably. In AranoDB documents are grouped into collections, that can be related to vertices or edges.

The last group is distributed processing frameworks that use property graphs [49, 50]. GraphX [49], which is an API of Apache Spark for graphs and graph-parallel computation, is an example of property graph usage. It extends the Spark RDD abstraction by introducing the resilient distributed Property Graph. SGraph, a part of GraphLab [50], is another scalable graph data structure, that derives from the property graph idea.

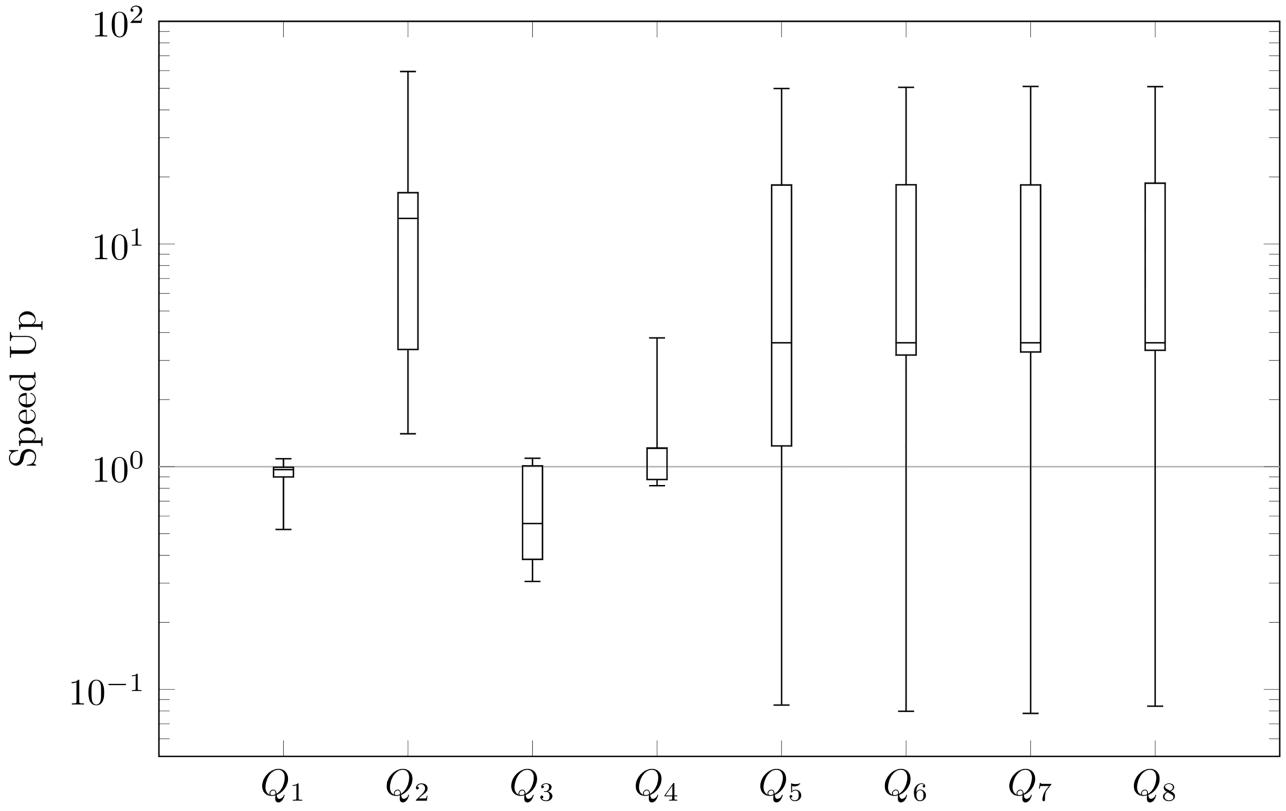

Fig 11. Speed-up of OrientDB SQL querying.

https://doi.org/10.1371/journal.pone.0191917.g011

5.2 Graph coloring

Graph coloring is known to be an NP-complete problem from the 70s [19, 51, 52]. Since then, the difficulty of this problem has contributed to proposed heuristic algorithms, hoping that the number of colors they use is near optimal [53–55]. Recently, the problem of minimum graph coloring is being better acquainted. Bellare *et al.* [56] show that minimum graph coloring cannot be approximated better than $|V|^{1/7-\epsilon}$ for every $\epsilon > 0$, unless $P = NP$. Moreover, Feige *et al.* [57] prove that if NP-problems cannot be solved by a randomized algorithm in polynomial time, the minimum graph coloring cannot be approximated better than $\Omega(|V^{1-\epsilon}|)$.

On the other hand, there are numerous heuristic algorithms for specialized graph coloring problems [58–61]. Qu *et al.* [58] propose a hybrid heuristic approach based on estimation distribution algorithms. This paper provides solution of acceptable quality for a number of optimisation problems and demonstrates the generality through experimental results for different variants of exam timetabling problems. FOO-PARTIALCOL is another approach presented in [59]. This method is based on tabu search [62]. A solution consists of k disjoint stable sets and a set of uncolored vertices. Yet another approaches are introduced in [60], which are based on Greedy algorithm [63]. In the first paper, Iterated Greedy algorithm is effective in graphs with n vertices partitioned into k as nearly equal sized sets as possible. In the second paper, the authors show heuristic methods to color vertices of a graph, which relies upon the comparison of the degrees and structures of graphs.

Several coloring algorithms occur in this context, depending on whether the matrix is a Jacobian [64, 65] or a Hessian [66, 67]. In [64], authors apply a column intersection graph-based formulation. In [65], authors propose formulation which is based on the concept of a

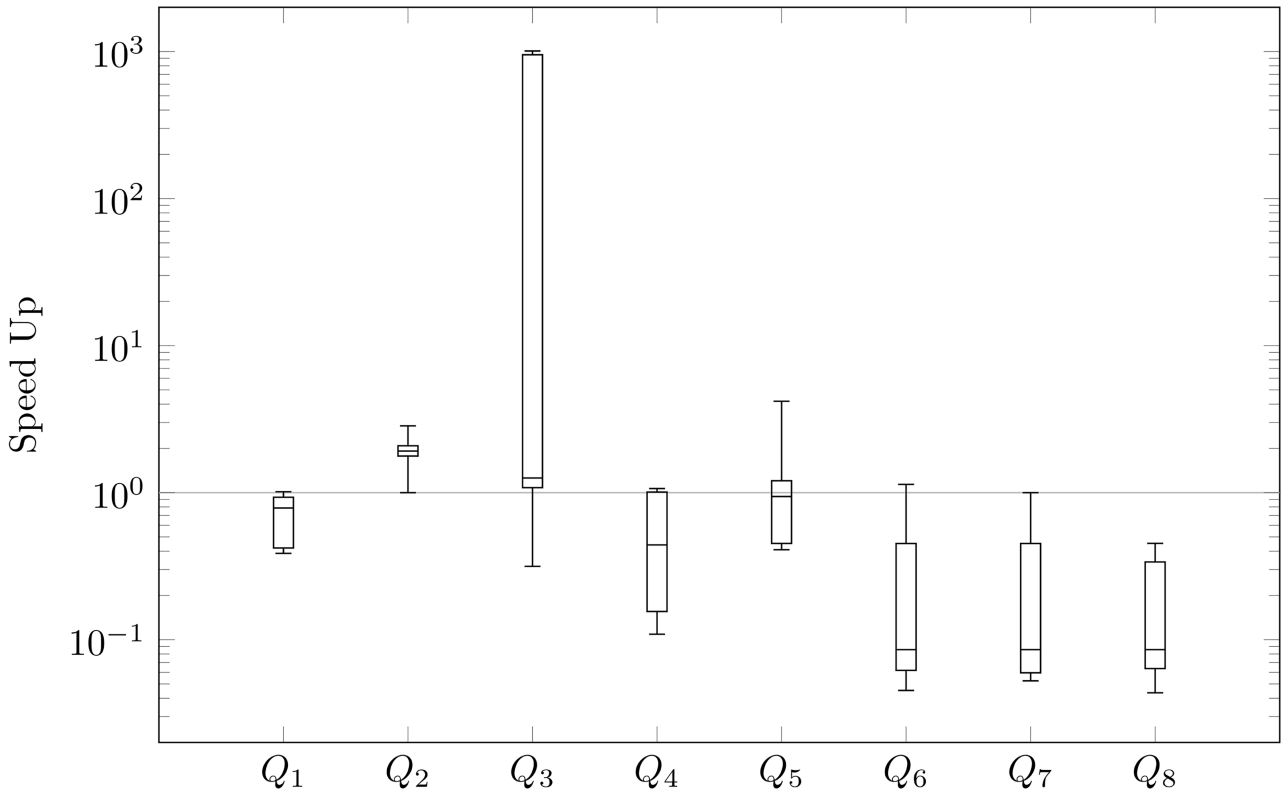

Fig 12. Speed-up of OrientDB Gremlin querying.

https://doi.org/10.1371/journal.pone.0191917.g012

Table 5. Distance-1 characteristics of serializations.

Format	Before [*B*]	After [*B*]	Ratio [%]
	DS_1		
GEXF	2234355	17154	130.25
GML	2247187	14369	156.39
GraphML	3233759	2385431	1.36
	DS_2		
GEXF	751755	11718	64.15
GML	832062	6789	122.56
GraphML	1177919	795143	1.48
	DS_3		
GEXF	115815	7432	15.58
GML	116634	8108	14.39
GraphML	150243	139259	1.08
	DS_4		
GEXF	684894	57395	11.93
GML	679809	48539	14.01
GraphML	911459	835754	1.09
	DS_5		
GEXF	928576	8257	112.46
GML	1029485	9008	114.29
GraphML	256936	154732	1.66

https://doi.org/10.1371/journal.pone.0191917.t005

Table 6. Distance-2 characteristics of serializations.

Format	Before [B]	After [B]	Ratio [%]
	DS_1		
GEXF	466185	462825	1.01
GML	470181	521293	0.90
GraphML	650897	541243	1.20
	DS_2		
GEXF	4730709	1391227	3.40
GML	4769025	1344635	3.55
GraphML	6466636	5007073	1.29
	DS_3		
GEXF	559304	553351	1.01
GML	564097	637539	0.88
GraphML	781198	637506	1.23
	DS_4		
GEXF	29082	9872	2.95
GML	29331	10519	2.79
GraphML	40604	24764	1.64
	DS_5		
GEXF	306051	130496	2.35
GML	318587	117081	2.72
GraphML	503349	292093	1.72

https://doi.org/10.1371/journal.pone.0191917.t006

consistent row-column partition, in which the entire set of rows and columns is partitioned into two respective sets of groups. Vertices that remain uncolored at the end of the algorithm form an independent set in the graph and can be assigned a neutral color zero. Coleman *et al.* [66] propose a model that exploits symmetry. This model is called *path coloring* and requires that every pair of adjacent vertices get distinct colors, and every path on four vertices uses at least three colors. McCormick [67] introduces a graph coloring model for the computation. The model uses the adjacency graph representation of the underlying symmetric matrix and requires that in every path u, v, w in the graph, vertices u, v, and w receive distinct colors.

5.3 Duplicate detection and deduplication

There are numerous approaches to duplication and similarity detection in general [68–76]. We distinguish probabilistic approaches [68, 69], supervised machine learning approaches [71, 72, 75] and unsupervised machine learning approaches [70, 73, 74, 76]. There are also solutions for duplication and similarity detection in graph data [77–80]. Dong *et al.* [77] present duplicate detection in a scenario where relationships between publications, persons, etc., form a graph. At each iteration, the first pair in the priority queue is retrieved, compared, and classified as nonduplicate or duplicate. The presented algorithm gradually enriches references by merging attribute values. Kalashnikov *et al.* [78] propose a domain-independent data cleaning approach for a graph of entities. Presented algorithm uses clustering techiques. Yin *et al.* [79] present linkage-based clustering, in which the similarity between two objects is measured based on the similarities between the objects linked to them. Bhattachary *et al.* [80] propose another algorithm that evaluates similarities of candidate pairs at each iterative step, and selects the most similar pair at each iteration. An algorithm augments a general class of attribute similarity measures with relational similarity among the entities. Duplicates are merged

together before the next iteration, so that in effect clusters of candidates are compared. This merge updates the reference graph and the priority queue.

On the other hand, we distinguish among approaches for deduplication: active learning methods [81, 82], clustering methods [83], and graph algorithms [84]. Sarawagi *et al.* [81] show how machine learning techiques could be applied in the elimination of redundant data where training data were available. Georgescu *et al.* [82] propose approach for deduplication with the main advantage of using crowdsourcing as a training and feedback mechanism. Culotta *et al.* [83] propose a conditional random field model of duplicate removal that captures these relational dependencies, and then employ a relational partitioning algorithm to jointly deduplicate data. Zhou *et al.* [84] show how to reduce the number of slow synchronization operations needed in parallel graph search.

6 Conclusions and future work

Reducing vertices is an important issue in graph databases. In this article, we outlined a way of reducing nodes of a property graph with the use of a graph vertex coloring method. We also presented works from the property graph coloring research area. Finally, we presented experiments, that showed great potential for the presented approaches.

The proposed approach is destined to evolve and include a wider set of coloring methods in its future versions. We showed that when reducing the number of vertices with one property, we can significantly increase the efficiency of working with graph databases that store RDF data, where the average number of speed-up is 24.99 times. However, the speed-up is significantly dependent on the choice of a graph database and a coloring method. Our initial research showed that we obtain large differences in the speed-up, even if we only focus on a single request in different graph databases.

Therefore, as part of future work, we will consider various algorithms for coloring graphs to find the ones that guarantee the best acceleration in the known graph databases. Furthermore, we will try to indicate which ones are best suited to each database. These studies will require not only coloring algorithms with the best approximation, but also heuristics that do not have constant approximations. Moreover, future work will focus on property graph partitions.

Author Contributions

Conceptualization: Dominik Tomaszuk, Karol Pąk.

Investigation: Dominik Tomaszuk.

Resources: Dominik Tomaszuk.

Software: Dominik Tomaszuk, Karol Pąk.

Visualization: Dominik Tomaszuk.

Writing – original draft: Dominik Tomaszuk, Karol Pąk.

References

1. Rodriguez MA, Neubauer P. Constructions from dots and lines. Bulletin of the American Society for Information Science and Technology. 2010; 36(6):35–41. https://doi.org/10.1002/bult.2010.1720360610
2. Gebremedhin AH, Manne F, Pothen A. What Color Is Your Jacobian? Graph Coloring for Computing Derivatives. SIAM Review. 2005; 47(4):629–705. https://doi.org/10.1137/S0036144504444711
3. Gebremedhin AH, Tarafdar A, Manne F, Pothen A. New Acyclic and Star Coloring Algorithms with Application to Computing Hessians. SIAM Journal on Scientific Computing. 2007; 29(3):1042–1072. https://doi.org/10.1137/050639879

4. Bornea MA, Dolby J, Kementsietsidis A, Srinivas K, Dantressangle P, Udrea O, et al. Building an Efficient RDF Store over a Relational Database. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. SIGMOD'13. New York, NY, USA: ACM; 2013. p. 121–132.
5. Lee K, Liu L. Scaling Queries over Big RDF Graphs with Semantic Hash Partitioning. Proc VLDB Endow. 2013; 6(14):1894–1905. https://doi.org/10.14778/2556549.2556571
6. Tomaszuk D, Skonieczny Ł, Wood D. In: Kozielski S, Mrozek D, Kasprowski P, Małysiak-Mrozek B, Kostrzewa D, editors. RDF Graph Partitions: A Brief Survey. Cham: Springer International Publishing; 2015. p. 256–264.
7. Cyganiak R, Wood D, Lanthaler M. RDF 1.1 Concepts and Abstract Syntax. World Wide Web Consortium; 2014.
8. Berners-Lee T, Hendler J, Lassila O. The Semantic Web. Scientific American. 2001; 284(5):34–43. https://doi.org/10.1038/scientificamerican0501-34
9. Karypis G, Kumar V. Multilevelk-way Partitioning Scheme for Irregular Graphs. J Parallel Distrib Comput. 1998; 48(1):96–129. https://doi.org/10.1006/jpdc.1997.1404
10. Gebremedhin AH, Nguyen D, Patwary MMA, Pothen A. ColPack: Software for Graph Coloring and Related Problems in Scientific Computing. ACM Trans Math Softw. 2013; 40(1):1:1–1:31. https://doi.org/10.1145/2513109.2513110
11. Hernández D, Hogan A, Riveros C, Rojas C, Zerega E. Querying Wikidata: Comparing SPARQL, Relational and Graph Databases. In: Groth, P, Simperl, E, Gray, A, Sabou, M, Krötzsch, M, Lecue, F, et al., editors. The Semantic Web—ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II. Cham: Springer International Publishing; 2016. p. 88–103.
12. Alocci D, Julien M, Oliver H, T BJ, P CM, Frederique L. Property Graph vs RDF Triple Store: A Comparison on Glycan Substructure Search. PLOS ONE. 2015; 10(12):1–17. https://doi.org/10.1371/journal.pone.0144578
13. Angles R, Prat-Pérez A, Dominguez-Sal D, Larriba-Pey JL. Benchmarking Database Systems for Social Network Applications. In: First International Workshop on Graph Data Management Experiences and Systems. GRADES'13. New York, NY, USA: ACM; 2013. p. 15:1–15:7.
14. Robinson I, Webber J, Eifrem E. Graph Databases. O'Reilly Media, Inc.; 2013.
15. Tomaszuk D. RDF Data in Property Graph Model. In: Garoufallou E, Subirats Coll I, Stellato A, Greenberg J, editors. Metadata and Semantics Research: 10th International Conference, MTSR 2016, Göttingen, Germany, November 22–25, 2016, Proceedings. Cham: Springer International Publishing; 2016. p. 104–115.
16. Auer S, Bizer C, Kobilarov G, Lehmann J, Cyganiak R, Ives Z. DBpedia: A Nucleus for a Web of Open Data. In: Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference. ISWC'07/ASWC'07. Berlin, Heidelberg: Springer-Verlag; 2007. p. 722–735.
17. Sperberg-McQueen M, Thompson H, Peterson D, Malhotra A, Biron PV, Gao S. W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. World Wide Web Consortium; 2012.
18. Adida B, Birbeck M, McCarron S, Herman I. RDFa Core 1.1—Third Edition: Syntax and processing rules for embedding RDF through attributes. World Wide Web Consortium; 2015.
19. Garey MR, Johnson DS. Computers and Intractability: A Guide to the Theory of NP-Completeness. New York, NY, USA: W. H. Freeman & Co.; 1979.
20. Isele R, Umbrich J, Bizer C, Harth A. LDspider: An Open-source Crawling Framework for the Web of Linked Data. In: Proceedings of the 2010 International Conference on Posters & Demonstrations Track—Volume 658. ISWC-PD'10. Aachen, Germany, Germany: CEUR-WS.org; 2010. p. 29–32.
21. Bizer C, Lehmann J, Kobilarov G, Auer S, Becker C, Cyganiak R, et al. DBpedia—A crystallization point for the Web of Data. Web Semantics: Science, Services and Agents on the World Wide Web. 2009; 7(3):154–165. https://doi.org/10.1016/j.websem.2009.07.002
22. Vrandecic D, Krötzsch M. Wikidata: A Free Collaborative Knowledgebase. Commun ACM. 2014; 57(10):78–85. https://doi.org/10.1145/2629489
23. Bizer C, Schultz A. The Berlin SPARQL Benchmark. Int J Semantic Web Inf Syst. 2009; 5(2):1–24.
24. Brickley D, Miller L. FOAF Vocabulary Specification 0.99; 2014.
25. Patel-Schneider PF, Motik B. OWL 2 Web Ontology Language: Mapping to RDF Graphs (Second Edition). World Wide Web Consortium; 2012.
26. Deutsch LP. DEFLATE Compressed Data Format Specification version 1.3. Internet Engineering Task Force; 1996. RFC1951.
27. Duda J, Tahboub K, Gadgil NJ, Delp EJ. The use of asymmetric numeral systems as an accurate replacement for huffman coding. In: 2015 Picture Coding Symposium (PCS). IEEE; 2015. p. 65–69.

28. Alakuijala J, Szabadka Z. Brotli Compressed Data Format. Internet Engineering Task Force; 2016. RFC7932.
29. Jouili S, Vansteenberghe V. An empirical comparison of graph databases. In: Social Computing (SocialCom), 2013 International Conference on. IEEE; 2013. p. 708–715.
30. Lal M. Neo4J Graph Data Modeling. Packt Publishing; 2015.
31. Tesoriero C. Getting Started with OrientDB. Packt Publishing Ltd; 2013.
32. Fielding RT, Reschke J. Hypertext transfer protocol (HTTP/1.1): Semantics and content. 2014;.
33. Fielding RT. REST: architectural styles and the design of network-based software architectures. University of California. Irvine; 2000.
34. Hu Y. Efficient, high-quality force-directed graph drawing. Mathematica Journal. 2005; 10(1):37–71.
35. Fruchterman TM, Reingold EM. Graph drawing by force-directed placement. Software: Practice and experience. 1991; 21(11):1129–1164.
36. Merkel D. Docker: Lightweight Linux Containers for Consistent Development and Deployment. Linux J. 2014; 2014(239).
37. Rodriguez MA. The Gremlin Graph Traversal Machine and Language. CoRR. 2015.
38. Marton J, Szárnyas G, Varró D. Formalising opencypher Graph Queries in Relational Algebra. CoRR. 2017.
39. Himsolt M. GML: A portable graph file format; 1997. Available from: http://www.fmi.uni-passau.de/graphlet/gml/gml-tr.html.
40. Brandes U, Eiglsperger M, Herman I, Himsolt M, Marshall MS. GraphML Progress Report Structural Layer Proposal. In: Mutzel P, Jünger M, Leipert S, editors. Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers. Berlin, Heidelberg: Springer Berlin Heidelberg; 2002. p. 501–512.
41. Hartig O. Reconciliation of RDF* and Property Graphs. CoRR. 2014.
42. Schätzle A, Przyjaciel-Zablocki M, Berberich T, Lausen G. S2X: Graph-Parallel Querying of RDF with GraphX. In: Wang F, Luo G, Weng C, Khan A, Mitra P, Yu C, editors. Biomedical Data Management and Graph Online Querying: VLDB 2015 Workshops, Big-O(Q) and DMAH, Waikoloa, HI, USA, August 31—September 4, 2015, Revised Selected Papers. Cham: Springer International Publishing; 2016. p. 155–168.
43. Batarfi O, Elshawi R, Fayoumi A, Barnawi A, Sakr S. A distributed query execution engine of big attributed graphs. SpringerPlus. 2016; 5(1):665. https://doi.org/10.1186/s40064-016-2251-0 PMID: 27350905
44. Hartig O, Thompson B. Foundations of an Alternative Approach to Reification in RDF. CoRR. 2014.
45. Martínez-Bazan N, Muntés-Mulero V, Gómez-Villamor S, Nin J, Sánchez-Martínez MA, Larriba-Pey JL. Dex: High-performance Exploration on Large Graphs for Information Retrieval. In: Proceedings of the Sixteenth ACM Conference on Conference on Information and Knowledge Management. CIKM'07. ACM; 2007. p. 573–582.
46. Das S, Srinivasan J, Perry M, Chong EI, Banerjee J. A Tale of Two Graphs: Property Graphs as RDF in Oracle. In: Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24–28, 2014.; 2014. p. 762–773.
47. Modoni GE, Sacco M, Terkaj W. A survey of RDF store solutions. In: 2014 International Conference on Engineering, Technology and Innovation (ICE). IEEE; 2014. p. 1–7.
48. Dohmen L, Klamma P, Celler F. Algorithms for large networks in the NoSQL database Arangodb. Bachelors thesis, RWTH Aachen, Aachen; 2012.
49. Xin RS, Gonzalez JE, Franklin MJ, Stoica I. GraphX: A Resilient Distributed Graph System on Spark. In: First International Workshop on Graph Data Management Experiences and Systems. GRADES'13. New York, NY, USA: ACM; 2013. p. 2:1–2:6.
50. Low Y, Bickson D, Gonzalez J, Guestrin C, Kyrola A, Hellerstein JM. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. Proc VLDB Endow. 2012; 5(8):716–727. https://doi.org/10.14778/2212351.2212354
51. Karp RM. Reducibility among Combinatorial Problems. In: Miller RE, Thatcher JW, Bohlinger JD, editors. Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department. Boston, MA: Springer US; 1972. p. 85–103.
52. Stockmeyer L. Planar 3-colorability is Polynomial Complete. SIGACT News. 1973; 5(3):19–25. https://doi.org/10.1145/1008293.1008294

53. Wood D. A technique for colouring a graph applicable to large scale timetabling problems. The Computer Journal. 1969; 12(4):317–319. https://doi.org/10.1093/comjnl/12.4.317

54. Welsh DJ, Powell MB. An upper bound for the chromatic number of a graph and its application to timetabling problems. The Computer Journal. 1967; 10(1):85–86. https://doi.org/10.1093/comjnl/10.1.85

55. Matula DW, Marble G, Isaacson JD. Graph coloring algorithms. Graph theory and computing. 1972; p. 109–122.

56. Bellare M, Goldreich O, Sudan M. Free Bits, PCPs, and Nonapproximability—Towards Tight Results. SIAM J Comput. 1998; 27(3):804–915. https://doi.org/10.1137/S0097539796302531

57. Feige U, Kilian J. Zero Knowledge and the Chromatic Number. J Comput Syst Sci. 1998; 57(2):187–199. https://doi.org/10.1006/jcss.1998.1587

58. Qu R, Pham N, Bai R, Kendall G. Hybridising heuristics within an estimation distribution algorithm for examination timetabling. Applied Intelligence. 2015; 42(4):679–693. https://doi.org/10.1007/s10489-014-0615-0

59. Blöchliger I, Zufferey N. A graph coloring heuristic using partial solutions and a reactive tabu scheme. Computers & Operations Research. 2008; 35(3):960–975. https://doi.org/10.1016/j.cor.2006.05.014

60. Culberson J. Iterated greedy graph coloring and the difficulty landscape. Edmonton, Alberta, Canada: University of Alberta; 1992.

61. Sen Sarma S, Bandyopadhyay S. Some sequential graph colouring algorithms. International Journal of Electronics Theoretical and Experimental. 1989; 67(2):187–199. https://doi.org/10.1080/00207218908921070

62. Hansen P. The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming. In: Proceedings of the Congress on Numerical Methods in Combinatorial Optimization. Capri, Italy; 1986.

63. Brélaz D. New methods to color the vertices of a graph. Communications of the ACM. 1979; 22(4):251–256. https://doi.org/10.1145/359094.359101

64. Coleman TF, Garbow BS, More JJ. Software for Estimating Sparse Jacobian Matrices. ACM Trans Math Softw. 1984; 10(3):329–345.

65. Hossain AKMS, Steihaug T. Computing a sparse Jacobian matrix by rows and columns. Optimization Methods and Software. 1998; 10(1):33–48. https://doi.org/10.1080/10556789808805700

66. Coleman TF, Moré JJ. Estimation of sparse hessian matrices and graph coloring problems. Mathematical Programming. 1984; 28(3):243–270. https://doi.org/10.1007/BF02612334

67. McCormick ST. Optimal approximation of sparse hessians and its equivalence to a graph coloring problem. Mathematical Programming. 1983; 26(2):153–171. https://doi.org/10.1007/BF02592052

68. Newcombe HB, Kennedy JM, Axford S, James AP. Automatic linkage of vital records. Science. 1959; 130(3381):954–959. https://doi.org/10.1126/science.130.3381.954 PMID: 14426783

69. Fellegi IP, Sunter AB. A Theory for Record Linkage. Journal of the American Statistical Association. 1969; 64(328):1183–1210. https://doi.org/10.1080/01621459.1969.10501049

70. Verykios VS, Elmagarmid AK, Houstis EN. Automating the approximate record-matching process. Information Sciences. 2000; 126(1):83–98. https://doi.org/10.1016/S0020-0255(00)00013-X

71. Cohen WW, Richman J. Learning to Match and Cluster Large High-dimensional Data Sets for Data Integration. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD'02. New York, NY, USA: ACM; 2002. p. 475–480.

72. Bansal N, Blum A, Chawla S. Correlation Clustering. Machine Learning. 2004; 56(1):89–113. https://doi.org/10.1023/B:MACH.0000033116.57574.95

73. Ravikumar P, Cohen WW. A Hierarchical Graphical Model for Record Linkage. In: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence. UAI'04. Arlington, Virginia, United States: AUAI Press; 2004. p. 454–461.

74. Bhattacharya I, Getoor L. A Latent Dirichlet Model for Unsupervised Entity Resolution. In: Proceedings of the 2006 SIAM International Conference on Data Mining. SIAM;. p. 47–58.

75. Bayardo RJ, Ma Y, Srikant R. Scaling Up All Pairs Similarity Search. In: Proceedings of the 16th International Conference on World Wide Web. WWW'07. New York, NY, USA: ACM; 2007. p. 131–140.

76. Hassanzadeh O, Chiang F, Lee HC, Miller RJ. Framework for Evaluating Clustering Algorithms in Duplicate Detection. Proc VLDB Endow. 2009; 2(1):1282–1293. https://doi.org/10.14778/1687627.1687771

77. Dong X, Halevy A, Madhavan J. Reference Reconciliation in Complex Information Spaces. In: Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data. SIGMOD'05. New York, NY, USA: ACM; 2005. p. 85–96.

78. Kalashnikov DV, Mehrotra S. Domain-independent Data Cleaning via Analysis of Entity-relationship Graph. ACM Trans Database Syst. 2006; 31(2):716–767. https://doi.org/10.1145/1138394.1138401

79. Yin X, Han J, Yu PS. LinkClus: Efficient Clustering via Heterogeneous Semantic Links. In: Proceedings of the 32Nd International Conference on Very Large Data Bases. VLDB'06. VLDB Endowment; 2006. p. 427–438.

80. Bhattacharya I, Getoor L. Collective Entity Resolution in Relational Data. ACM Trans Knowl Discov Data. 2007; 1(1). https://doi.org/10.1145/1217299.1217304

81. Sarawagi S, Bhamidipaty A. Interactive Deduplication Using Active Learning. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD'02. New York, NY, USA: ACM; 2002. p. 269–278.

82. Georgescu M, Pham DD, Firan CS, Nejdl W, Gaugaz J. Map to Humans and Reduce Error: Crowdsourcing for Deduplication Applied to Digital Libraries. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management. CIKM'12. New York, NY, USA: ACM; 2012. p. 1970–1974.

83. Culotta A, McCallum A. Joint Deduplication of Multiple Record Types in Relational Data. In: Proceedings of the 14th ACM International Conference on Information and Knowledge Management. CIKM'05. New York, NY, USA: ACM; 2005. p. 257–258.

84. Zhou R, Hansen EA. Parallel Structured Duplicate Detection. In: Proceedings of the 22Nd National Conference on Artificial Intelligence—Volume 2. AAAI'07. AAAI Press; 2007. p. 1217–1223.