Quality Awareness over Graph Pattern Queries

Philippe Rigaux, Virginie Thion

Quality Awareness over Graph Pattern Queries

Philippe Rigaux
CNAM/CEDRIC,
Paris, France
Philippe.Rigaux@cnam.fr

Virginie Thion
Univ. Rennes 1/IRISA,
Lannion, France
Virginie.Thion@irisa.fr

ABSTRACT

We examine the problem of quality awareness when querying graph databases. According to quality annotations that denote quality problems appearing in data subgraphs (the annotations typically result from collaborative practices in the context of open data usage like e.g. users' feedbacks), we propose a notion of *quality aware (graph pattern) query* based on (usage-dependent) quality profiles. In this paper, we present the formal foundations of the approach. We also show how to simply extend a generic state-of-the-art algorithm for graph pattern queries evaluation in order to implement quality awareness at evaluation time and we study its complexity. We then expose implementation guidelines, supported by a proof-of-concept prototype based on the Neo4J graph database management system.

CCS CONCEPTS

•**Information systems → Query languages for non-relational engines; Evaluation of retrieval results;** Graph-based database models;

KEYWORDS

Graph Databases, Graph pattern query, Data Quality

Much work has been done about data quality management in *relational* databases (see eg. [11] and [10]). However, even though relational databases are still widely used, the need to handle *complex* data has led to the emergence of other types of data models. In the last few years, *graph databases* have started to attract a lot of attention in the database world (see e.g. [7], [16], [14], [25] and [6]). Their basic purpose is to manage networks of entities, the underlying data model of many open data applications like e.g. social networks, biological or bibliographic databases. This context raises new challenges in terms of data quality management.

Literature proposed a wide range of metrics that make it possible to measure data quality for existing data models including graph-based ones (see eg. [11] and [27]). These metrics are used to detect quality problems in data and to measure the quality level of data. But, assuming that data quality is known, a question still raises, that is to say "How to take quality information into account at query time?" This is the problem we tackle here.

We propose an extension of the graph database querying process that allows introducing quality awareness when querying data. Based on *quality annotations* that denote quality problems appearing in data subgraphs (the annotations may result either from an automatic evaluation of data quality, for instance by computing quality metrics defined in literature [17, 27], or from a human tagging process that is a typical collaborative practice in the context of open data usages [26, 5]) and a *quality vocabulary*, we propose a notion of *quality aware query* based on (usage-dependent) *quality profiles* defined according to the quality vocabulary.

The paper is organized as follows. In Section 1, we first present the considered data model, an extension of the attributed graph data model that makes is possible to model data, quality problems, a quality vocabulary and user quality profiles in the attributed graph formalism. We then propose in Section 2 a quality aware extension of the graph pattern query notion. In Section 3, we show how to extend a generic state-of-the-art algorithm for introducing quality awareness computation at query evaluation time, and we exhibit the cost of this extension. In Section 4, we briefly expose implementation guidelines supported by a proof-of-concept prototype that introduces quality awareness in the Neo4J Cypher query language. Related work is discussed in Section 5. At last, we recall the contributions and outline some perspectives in Section 6.

1 DATA MODEL

In a graph database management system, the schema is modeled as a graph (nodes are entities and edges are relations between entities), and data is handled through graph-oriented operations and type constructors [7]. Salient systems are AllegroGraph [1], InfiniteGraph [2], Neo4j [3] and Sparksee [4]. There are different models for graph databases [7], including the *attributed graph* (aka. *property graph*) where nodes and edges may embed data in *attributes*. This is the data model that we consider in this work.

In the following, we assume the existence of pairwise disjoint sets: sets of nodes $\mathcal{V}$ (nodes in data) and $\mathcal{V}_Q$ (nodes in the quality vocabulary), a set $\mathcal{E}$ of labels and a set $\mathcal{A}$ of attribute names (the domain an attribute $a \in \mathcal{A}$ is denoted by $Dom(a)$).

Data Graph. We first consider the notion of data graph. A *data graph* $\mathcal{G}$ is a tuple (V, R), where

- $V \subseteq \mathcal{V}$ is a finite set of nodes and
- $R = \{r_e \mid e \in \mathcal{E} \text{ and } r_e \subseteq V \times V\}$ is a set of labeled edges between nodes.

An *attributed graph* makes it possible to embed attributes (key-values pairs) in nodes and edges. It is modelled as a quadruple (V, R, ζ_V, ζ_R) where

- (V, R) is a data graph,
- $\zeta_V = \{\zeta_V^a \mid a \in \mathcal{A} \text{ and } \zeta_V^a : V \rightarrow \text{Dom}(a)\}$ assigns attribute values to nodes and

IDEAS 2017, Bristol, United Kingdom
 978-1-4503-5220-8…$15.00
DOI: 10.1145/3105831.3105871

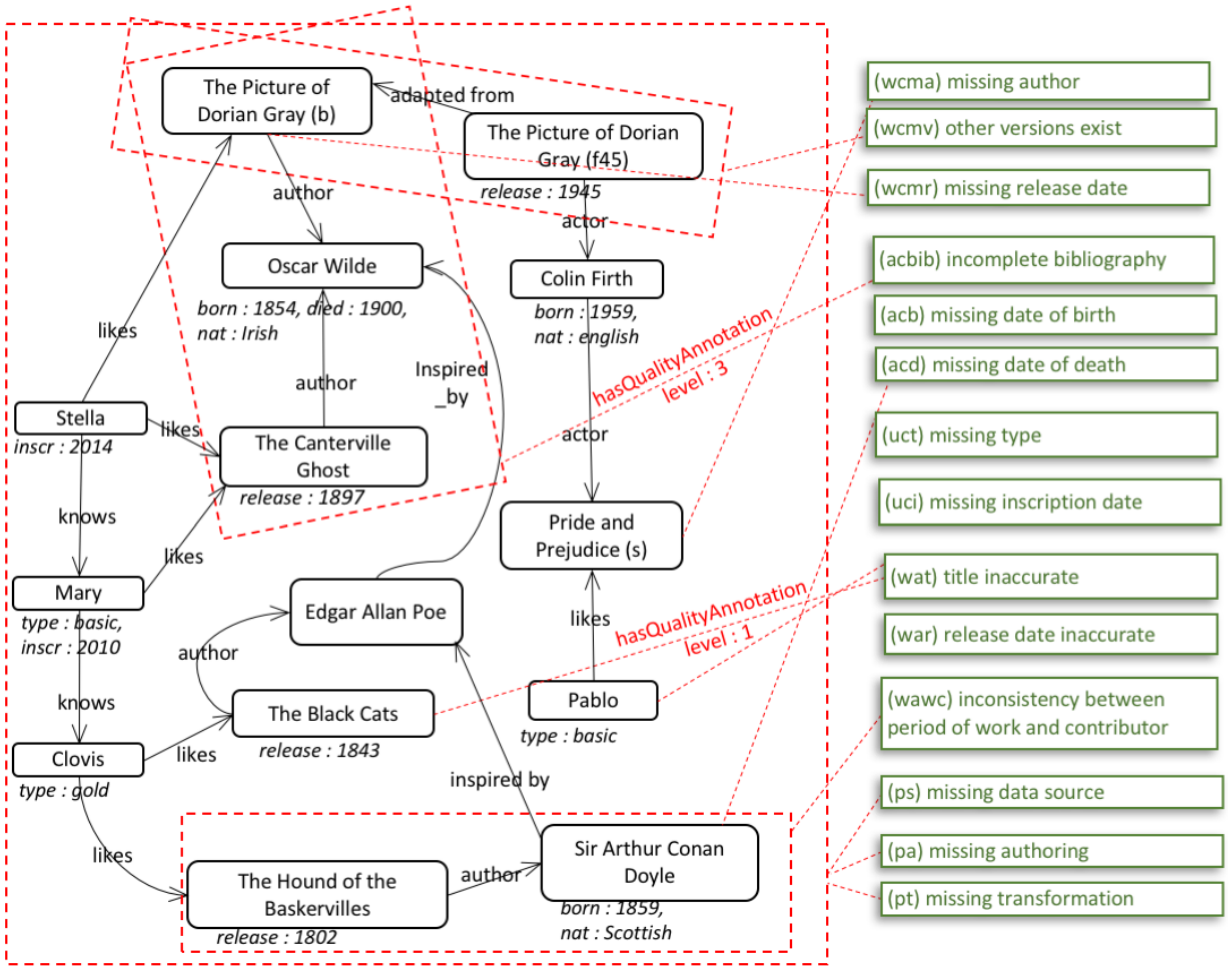

Figure 1: Data graph $\mathcal{G}$ (black) and association (red) of quality annotations (green)

- $\zeta_R = \{\zeta_r^a \mid a \in \mathcal{A}$ and $r \in R$ and $\zeta_r^a : Dom(r) \rightarrow \text{Dom}(a)\}$ assigns attribute values to edges.

Example 1.1. The left black part of Figure 1 is an example of data graph that models data of a social network information system dedicated to literature. It contains nodes denoting users, works of art and artists (authors and actors), and connections between nodes that we expect as being explicit enough for not to detail them. Attributes are written in italics. ⋄

We now consider how to attach quality annotations to such data, and how to define user quality profiles.

Quality annotations. The W3C proposes a quality vocabulary [13] that allows attaching quality measures and annotations to RDF data graphs. We consider an excerpt of this vocabulary by focusing on the *hasQualityAnnotation* relationship that makes it possible to attach quality annotations to data. Each annotation belongs to a quality *dimensions* [11] aiming to categorize quality measures of interest. Classical quality dimensions are *completeness* (the degree to which needed information is present in data), *accuracy* (the degree to which data are correct), *consistency* (the degree to which data respect integrity constraints and business rules) and *freshness* (the degree to which data are up-to-date).

We adapt this vocabulary to the attributed graph model and slightly generalize it. First, we extend it in order to consider a more general multi-level taxonomy. Second, as a quality annotation may reflect different level of seriousness of a problem, we also allow to attach a degree of seriousness to a *hasQualityAnnotation* relation. The quality vocabulary forms a taxonomy of terms.

Quality Vocabulary. A *quality vocabulary* $\mathcal{V}oc$ is denoted by a tree (V, r), where $V \subseteq \mathcal{V}_Q$ and $r \subseteq V \times V$ classifies elements of V. The leaves of $\mathcal{V}oc$, referred to as Problems($\mathcal{V}oc$), are annotations that may be attached to data. An annotation denotes a quality problem. r forms a taxonomy of terms over V. Internal nodes of $\mathcal{V}oc$ allow classifying the annotations.

We introduce the following shortcut notation: given a data graph $\mathcal{G}$, if p is a quality problem then $AnnotatedBy(p) \subseteq \mathcal{P}(\mathcal{G})$ denotes the subgraphs of $\mathcal{G}$ to which p is attached.

Example 1.2. ***(Continued)*** The figure 2 is an example of quality vocabulary with four classification levels (including the *Top* one).

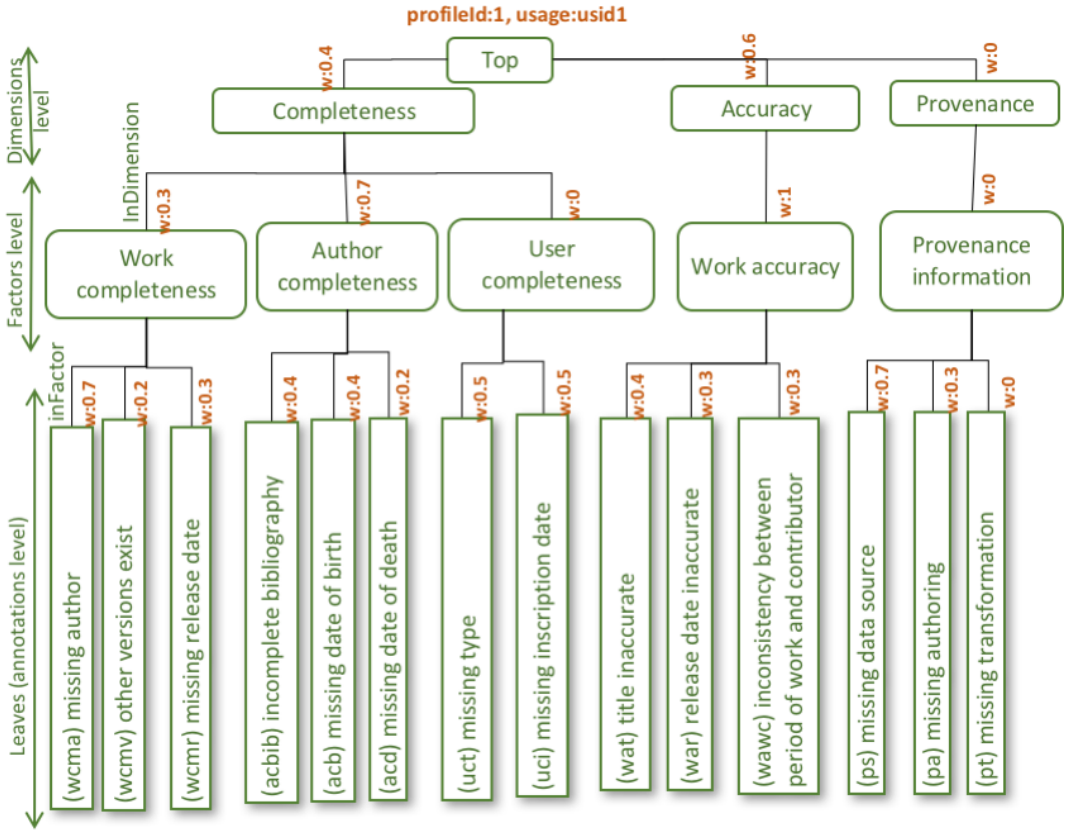

Figure 2: Quality taxonomy (green) and quality profile (orange)

In Figure 1, some values of the quality association (*hasQuality-Association* relationship) are given (dashed lines); some quality annotations defined in the quality vocabulary of Figure 2 are attached to data subgraphs. ⋄

We do not consider the problem of the quality problems assignment, which may be an automatic (e.g. like proposed in [17] where quality template queries allow to detect quality problems, or by computing metrics of literature [27]), or a human collaborative process (e.g. like proposed in [26, 5]).

Quality profiles. Data quality is defined according to *fitness for use* of data, meaning that it depends on the data *usage*. A user may be concerned by some quality problems for a specific usage, by some other problems for another one, and they can be completely different than those considered by another user. Then there is a clear need of personalized data access. Methods based on personalization consist in defining users' profiles and then using them in order to generate preferences incorporated in user queries [23].

According to a usage of data, we define a *quality profile* by attaching weights to edges of the quality vocabulary. A weight defines the degree of interest of a quality element (node of the vocabulary) for the usage. It is a vectorial profile based on the vocabulary. Several quality profiles (for different usages of data) should be defined.

Definition 1.3 (Syntax of a Quality Profile). A *(quality) profile* over a vocabulary $\mathcal{V}oc = (V, r)$ is defined by a total function $\mathrm{w} : r \to \mathbb{R}$ that specify, for each $e = (v_1, v_2)$ of r, the level of importance of v_2 (possibly relatively to its siblings), for its parent element v_1.

Example 1.4. ***(Continued)*** Weights joined to the vocabulary in Figure 2 constitute an example of profile, denoted by *profileId1* in the following. ⋄

Given this data model, our goal is then to extend basic graph pattern queries with a computation of quality scores for the answers, according to a given quality profile.

2 QUALITY AWARE QUERYING

A *graph pattern query* is classically defined as a graph where variables and conditions can occur. Let $\mathcal{V}_{nodes}$ and $\mathcal{V}_{lab}$ be distinct sets of node and edge variables respectively. A *graph pattern query* is a tuple of the form

$$(V', R', \sigma_n, \sigma_e)$$

where

- $V' \subseteq \mathcal{V} \cup \mathcal{V}_{nodes}$ denotes the nodes of the graph pattern (where variables can occur);
- $R' = \{r_e \mid e \in \mathcal{E} \cup \mathcal{V}_{lab}$ and $r_e \subseteq V' \times V'\}$ denotes the set of edges of the graph pattern (where variables can occur);

- σ_{nodes} (resp. σ_{lab}) denotes Boolean conditions over attribute values of the elements of V' (resp. R'), for instance n.nat = 'en' ∧ n.born > 1880, where n is an author node.

Example 2.1. ***(Continued)*** The figure 3 is a graph pattern query, denoted by $\mathcal{P}_{\text{Clovis}}$, that aims to retrieve works of art of authors inspired by authors of works that Clovis likes. ⋄

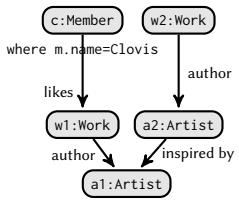

Figure 3: $\mathcal{P}_{\text{Clovis}}$

The answer $[\![\mathcal{P}]\!]_{\mathcal{G}}$ of a pattern query $\mathcal{P}$ over a graph $\mathcal{G}$ is the set of subgraphs $\{g \in \mathcal{P}(\mathcal{G}) \mid g$ "matches" $\mathcal{P}\}$. A subgraph g "matches" $\mathcal{P}$ iff there exists a homomorphism h from nodes and labels of $\mathcal{P}$ to g and each node $h(n)$ (resp. label $h(e)$) satisfies its associated conditions $\sigma_{nodes}(n)$ (resp. $\sigma_{lab}(e)$).

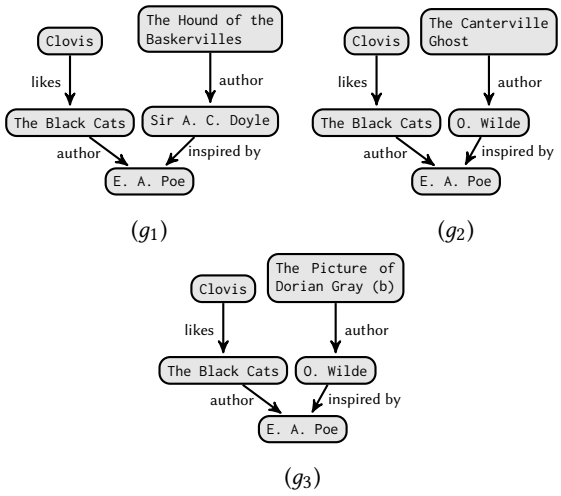

Figure 4: Answers of $\mathcal{P}_{\text{Clovis}}$ over $\mathcal{G}$

Example 2.2. ***(Continued)*** The figure 4 presents the answer of $\mathcal{P}_{\text{Clovis}}$ (Figure 3) over the graph $\mathcal{G}$ (Figure 1), which correspond to the three mappings

- m_1 = {c → Clovis, a1 → E. A. Poe, w1 → The Black Cats, a2 → Sir A. C.Doyle, w2 → The Hound of the Baskervilles},
- m_2 = {c → Clovis, a1 → E. A. Poe, w1 → The Black Cats, a2 → O. Wilde, w2 → The Canterville Ghost} and
- m_3 = {c → Clovis, a1 → E. A. Poe, w1 → The Black Cats, a2 → O. Wilde, w2 → The Picture of Dorian Gray (b)}. ⋄

We now turn to the extension of this concept in order to introduce quality awareness according to the considered data model (Section 1). We first present the syntax of a quality aware graph pattern query, and then its semantics.

In terms of syntax, the notion of *quality aware query* is an extension of the graph pattern query. Given a quality profile, it consists in defining elements of interest over the quality profile. This syntax is formally defined by the definitions 2.3.

Definition 2.3 (Quality Aware Query Syntax). Given a vocabulary $\mathcal{V}oc = (V, r)$, a *quality aware (graph pattern) query* is a tuple

$$(\mathcal{P}, prof, qtInterest)$$

where

- $\mathcal{P}$ is a graph pattern;
- *prof* is a quality profile defined over $\mathcal{V}oc$ and
- *qtInterest* is a subset of V.

Example 2.4. ***(Continued)*** An example of quality aware query is

$$Q_{qt} = (\mathcal{P}_{\text{Clovis}}, profileId1, \{\text{Completeness, Accuracy}\})$$

that aims to retrieve answers of $\mathcal{P}_{\text{Clovis}}$ with their alert score over the quality dimensions *Completeness* and *Accuracy* according to the profile *profileId1*. ⋄

In terms of semantics, intuitively, given a graph database, a quality vocabulary $\mathcal{V}oc$ and quality associations (*hasQualityAnnotation* relationship), the evaluation of a quality aware query ($\mathcal{P}$, *prof*, *qtInterest*) returns the set of answers of the pattern query $\mathcal{P}$, and, with each of the answers, an associated quality alert score calculated according to (i) the profile *prof*, (ii) the quality annotations attached to data, and (iii) restricted to *qtInterest*. The semantics are formally defined by the definitions 2.5.

Remark: For the score calculation, we propose below a simple definition of alert score based on the number and degree of seriousness of the quality problems attached to an answer, which can easily be extended.

Definition 2.5 (Quality Aware Query Semantics). Given a quality vocabulary $\mathcal{V}oc = (V, r)$, the *interpretation* $[\![Q]\!]_{\mathcal{G}}$ *of a quality aware query* $Q = (\mathcal{P}, prof, qtInterest)$ over a graph $\mathcal{G}$ is the largest set of pairs $(g, sc(g))$ such that $g \in [\![\mathcal{P}]\!]_{\mathcal{G}}$, and $sc(g)$ is a quality alert score of the form $\{m : sc(g, m) \mid m \in qtInterest\}$, recursively defined as follows (bottom-up computation in the quality profile tree):

(1) if m is an element of Problems($\mathcal{V}oc$) then $sc(g, m)$ is

$$\sum_{\substack{\{s \mid s \in AnnotatedBy(m) \\ \text{and } s \cap g \neq \text{null graph}\}}} hasQualityAnnotation(s, m)$$

meaning that the quality score cumulates degrees of seriousness of each parts of g to which e is associated with;

(2) else the score of m is a weighted average of its childs 'scores: $sc(g, m)$ is $\sum_{e=(m,v_2)\in r} w(e) \times sc(g, v_2)$, where w is the weigh defined over e by the profile *prof*.

Remark: Ordering the answers by quality score means performing a descendant ordering according to the alert level.

Example 2.6. ***(Continued)*** Let us consider the quality aware query Q_{qt} defined in Example 2.4. The answers of Q_{qt} over $\mathcal{G}$, denoted by $[\![Q_{qt}]\!]_{\mathcal{G}}$, are the subgraphs g_1, g_2 and g_3 of Figure 4 with the associated quality scores:
$[\![Q_{qt}]\!]_{\mathcal{G}} = \{$
$(g_1, \{Completeness : 0.14, Accuracy : 0.7\}),$
$(g_2, \{Completeness : 0.84, Accuracy : 0.4\}),$
$(g_3, \{Completeness : 0.84, Accuracy : 0.4\})\}$. $\diamond$

The higher the score is, the higher is the quality alert level for the answer on the dimension (so the more suspect is the quality of the answer).

Generalizability of the approach. Choosing another score calculation based on the vocabulary hierarchy (unweighted sum, max, quantifiers, etc) is straightforward. Extending the query syntax in order to allow regular path expressions on pattern edges is also trivial as it only impacts the definition of the graph pattern, independently from the quality awareness extension (for the sake of simplicity, this extension is not presented here but it is discussed in Section 3).

3 ALGORITHM AND COST

The cost of a graph pattern query evaluation depends on the form of the pattern. The data complexity over arbitrary graph pattern queries (conjunctive regular path queries including regular expressions with variables on edges) is NP-complete [8]. Tractable fragments are based on syntactical restriction of the pattern that either takes the form of a regular path query (the pattern is only a path connecting two nodes, defined by a regular expression) or more generally a conjunctive regular path query (a graph pattern where each edge is a regular path expression) possibly including specific navigational capabilities like eg. inverse traversal, nested regular expressions, or memory registers (see e.g. [8, 9]). For instance, in the case of a Conjunctive Regular Path Queries (CRPQ) Q, meaning patterns with variables allowed on nodes, and edges labelled in $REG(\mathcal{E})$, the cost of the evaluation is of the order $|\mathcal{G}|^{O(|Q|)}$ [24]. If the considered pattern is a 2RPQ (regular path queries with reverse traversal) L then the evaluation is in linear time $O(|\mathcal{G}| \cdot |L|)$ [8].

We deal here with quality aware query evaluation. In order to exhibit the additional cost incurred by quality evaluation and to provide algorithm that guides the implementation of the approach, we propose an extension of the generic graph pattern evaluation algorithm proposed in [18], which constitutes a common framework to other state-of-the-art algorithms including GraphQL, QuickSI, SPath and SwiftIndex (see [18] for the comprehensive list of studied algorithms). This algorithm is composed of a common skeleton that invokes subroutines whose implementation is specific to each of the state-of-the-art algorithms. Originally designed in order to compare some the existing algorithms, it can also be seen as a uniform vision that allow projecting in each of them. Without giving all the details of the generic algorithm, we can explain that given a pattern query Q and a data graph $\mathcal{G}$, the algorithm calculates $[\![Q]\!]_{\mathcal{G}}$ by a trial and error procedure, that recursively constructs each answer mapping by adding one edge of the answer at a time.

The extension for handling quality aware queries, based on the approach proposed in Sections 1 and 2, aims at calculating a quality alert score associated with each answer. This impacts the algorithm of [18] is several ways: (1) each answer mapping M has an associated quality alert score that defines a score $sc[M, p]$ for each element p of the quality taxonomy, (2) the SubgraphSearch subroutine is modified to include a quality score calculation when an answer is found (described in Algorithm 1).

Algorithm 1: SubgraphSearch additional routine

```
1 if |M| = |V(q)|                          // If M is complete and satisfying
2 then
    // Initialisation of the quality scores
3   Visited ← ∅;
4   foreach p ∈ Problems(Voc) do sc[M, p] = 0;
    // Calculation of the quality scores
5   foreach element (node or edge) e of M do
6     foreach association hasQualityAnnotation(g_in, p) s.t.
        e ∈ g_in and hasQualityAnnotation(g_in, p) ∉ Visited
        do
7       sc[M, p] ←
          sc[M, p] + hasQualityAnnotation(g_in, p);
8       Visited ←
          Visited ∪ {hasQualityAnnotation(g_in, p)};
9   report M and sc;
```

Finally, when reporting the answer in line 9, the score associated with each internal element of *qtInterest* is calculated thanks to a recursive bottom-up traversal of the quality profile, with a clearly negligible cost.

The additional cost then consists in calculating, for each subgraph answer, the score associated with each internal element of *qtInterest* by a recursive bottom-up traversal of the taxonomy, whose cost is in $O(\Sigma_{(g,s)\in[\![Q]\!]_{\mathcal{G}}}|N_g| + |E_g|) \cdot |\text{Problems}(\mathcal{V}oc)|)$ where N_g (resp. E_g) is the set of nodes (resp. edges) in the answer g. Assuming that the width of the taxonomy is smaller than $\mathcal{G}$, this cost is less than $O(|[\![Q]\!]_{\mathcal{G}}| \cdot |\mathcal{G}|^2)$.

In the case of a CRPQ, this additional cost is highly dominated by the cost of the evaluation without quality awareness, which is in $|\mathcal{G}|^{O(|Q|)}$. If the considered pattern is a 2RPQ then this cost seems reasonable w.r.t. a non-quality aware evaluation, which is in $O(|\mathcal{G}| \cdot |Q|)$ [8].

4 IMPLEMENTATION ISSUES

Implementation-wise, two architectures may be thought of. A first one consists in implementing a specific quality aware query evaluation engine. The advantage of this solution is that optimization techniques implemented directly in the query engine should make the system very efficient for query processing. The downside is that quality aware queries may not be evaluated by an independent engine that does not implement the quality aware functionality.

The second solution, chosen here, consists in using a possibly distant classical engine, combined with a dedicated add-on layer.

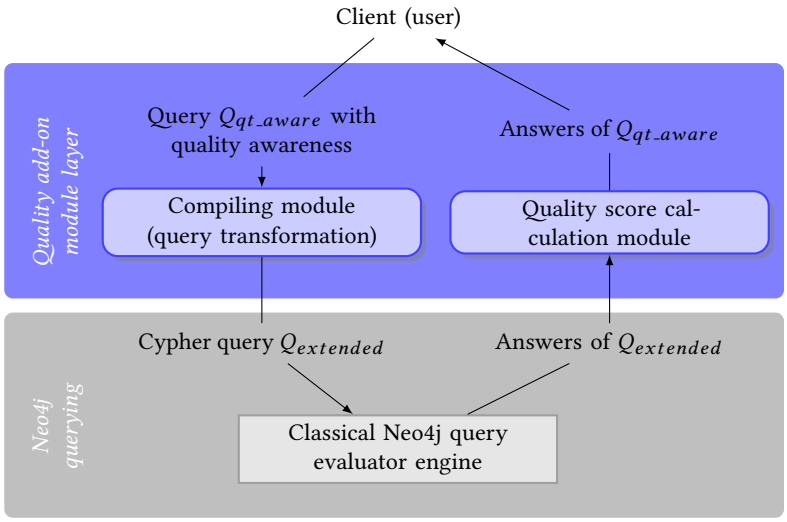

Figure 5: Architecture of TAMARI

The implementation relies on a query-rewriting derivation mechanism, carried out as a pre-processing and a post-processing steps.

As a proof-of-concept of the proposed approach, the open-source prototype TAMARI[1] (for Quality Alerts Management using RabbIthole) adds quality awareness to the Cypher language [19] for querying a Neo4j graph database [3]. TAMARI implements a quality add-on layer on top of a classical (non-quality aware) Cypher query engine, by extending the RabbitHole console [22]. The architecture of TAMARI is depicted in Figure 5.

The add-on layer is composed of two modules. A *Compiling module* transforms the graph pattern Cypher query Q_{qt_aware} into an extended one $Q_{extended}$ that retrieves all the needed information not only concerning the answers but also from the user profile, the quality vocabulary and the association of the vocabulary to the answers. The extended query is then sent to the (classical) Neo4j engine. Based on the answers of $Q_{extended}$, a *Quality score calculation module* calculates the quality alert scores associated with each answer of Q_{qt_aware}[2].

The figure 6 is a screenshot of the TAMARI graphical user interface, after the evaluation of the running example query, expressed in the Cypher language, given in Query 1.

```
1 QTAWARE profileId1, Completeness, Accuracy
2 MATCH
3   (c)−[l:likes]−>(w1:Book), (w1)−[:author]−>(a1),
4   (a2)−[:inspired_by]−>(a1), (w2)−[:author]−>(a2)
5 WHERE c.name='Clovis'
6 RETURN c, w1, a1, a2, w2
```

Query 1: Quality aware Cypher query (Q_{qt})

[1] TAMARI is available at www-shaman.irisa.fr/tamari.

[2] Note that, in terms of expressivity, this calculation cannot be expressed in the extended query. A calculator module is then needed.

For a quality aware graph pattern query $(\mathcal{P}, prof, qtInterest)$ (see Definition 2.3), the considered profile $prof$ and quality elements of interest $qtInterest$ are defined in a QTAWARE clause (line 1 in Query 1) that extends the Cypher language, and the pattern $\mathcal{P}$ is defined according the Cypher syntax in the following of the query (in lines 2 to 5 in the MATCH/WHERE clauses of Query 1).

The result of the evaluation of Query 1 is given in the upper frame of the user interface in Figure 6, under the form of a table that presents three answers (mappings of the query variables to elements of the data graph). Not surprisingly, the answers appearing in Figure 6 are those given in Example 2.6, with their expected quality alert score.

5 RELATED WORK

On one hand, the literature proposes a large amount of contributions concerning data quality management [11] and more specifically over graph databases [27, 13]. First one has to note that most existing frameworks propose to attach quality measurements, and not quality problems, to data. Yet, reasoning in terms of problems is more intuitive for users and convenient for collaborative quality assessment in which users may "tag" data with observed quality problems at usage time. The collaborative evaluation of the quality in graph data, which implies considering data annotated with quality problems, has only recently been studied. In [5], the authors propose a framework based on crowdsourcing in order to attach quality problems, classified by categories, to RDF graph data, but the querying process of such data is not tackled.

On the other hand, a large amount of contributions have been proposed in order to introduce preferences in queries [23], but few works consider preference queries for graph data. Most of the contributions consider the RDF model [20]. To our knowledge, [15] is the only contribution that considers preferences over the attributed graph model, based on a pareto order over paths connecting nodes. None of these works considers data annotated with quality problems and quality preferences based on it.

Figure 6: Screenshot of the TAMARI prototype

Another close work is [21] in which authors define a Cypher extension that makes it possible to express preferences, based on fuzzy logic, in graph pattern queries in order to allow a user to express flexible patterns in the Neo4j Cypher language. In this work, data does not embed quality information, and quality awareness of queries is not studied.

6 CONCLUSION

We study the problem of quality awareness in graph databases. Based on quality annotations that denote quality problems appearing in data subgraphs, and a quality vocabulary, we propose a notion of quality aware query based on (usage-dependent) quality profiles defined as specializations of the vocabulary. We extend a generic state-of-the-art algorithm for introducing quality awareness computation at query evaluation time, and we exhibit the cost of this extension. We also discuss implementation issues. A prototype implementing the framework in Neo4j supports the proposed approach.

The present work opens many perspectives, including its generalization to more complex quality vocabularies [11] and the definition of more complex and flexible quality-based preference queries [23, 20, 21], for instance considering skyline queries or fuzzy quality preferences. Experimentations (benchmarking and users feedback analysis) of such approaches on possibly large databases have to be performed. The framework is about to be experimented in the context of digital score libraries [12].

7 ACKNOWLEDGMENTS

This work has been funded by the French National Center for Scientific Research (CNRS) under the *défi Mastodons GioQoso* and the French DGE (Direction Générale des Entreprises) under the project ODIN (Open Data INtelligence).

REFERENCES

[1] AllegroGraph web site. franz.com/agraph/allegrograph.
[2] InfiniteGraph web site. www.objectivity.com/infinitegraph.
[3] Neo4j web site. www.neo4j.org.
[4] Sparksee web site. sparsity-technologies.com.
[5] M. Acosta, A. Zaveri, E. Simperl, D. Kontokostas, S. Auer, and J. Lehmann. Crowdsourcing linked data quality assessment. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 260–276, 2013.
[6] R. Angles. A comparison of current graph database models. In *Proceedings of International Conference on Data Engineering (ICDE) Workshops*, pages 171–177, 2012.
[7] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Computing Surveys (CSUR)*, 40(1):1–39, 2008.
[8] P. Barceló. Querying graph databases. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 175–188, 2013.
[9] P. Barceló, L. Libkin, and J. L. Reutter. Querying regular graph patterns. *J. ACM*, 61(1):8:1–8:54, 2014.
[10] C. Batini, C. Cappiello, C. Francalanci, and A. Maurino. Methodologies for data quality assessment and improvement. *ACM Computing Surveys (CSUR)*, 41(3), 2009.
[11] C. Batini and M. Scannapieco. *Data and Information Quality: Dimensions, Principles and Techniques*. Springer, 2016.
[12] V. Besson, M. Gurrieri, P. Rigaux, A. Tacaille, and V. Thion. A Methodology for Quality Assessment in Collaborative Score Libraries. In *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, 2016.
[13] J. Debattista, M. Dekkers, C. Guéret, D. Lee, N. Mihindukulasooriya, and A. Zaveri. Data on the web best practices: Data quality vocabulary. W3C Working Group, december 2016.
[14] D. Dominguez-Sal, P. Urbón-Bayes, A. Giménez-Vañó, S. Gómez-Villamor, N. Martínez-Bazan, and J.-L. Larriba-Pey. Survey of Graph Database Performance on the HPC Scalable Graph Analysis Benchmark. In *Proceedings of the International Conference on Web-Age Information Management (WAIM) Workshops*, pages 37–48, 2010.
[15] V. Fionda and G. Pirro'. Querying graphs with preferences. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 929–938. ACM, 2013.

[16] R. Giugno and D. Shasha. GraphGrep: A Fast and Universal Method for Querying Graphs. In *Proceedings of the International Conference on Pattern Recognition (IPCR)*, pages 112–115, 2002.
[17] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, and A. Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the International World Wide Web Conferences (WWW)*, pages 747–758, 2014.
[18] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee. An in-depth comparison of subgraph isomorphism algorithms in graph databases. *Proceedings of the Very Large Database Endowment (PVLDB)*, 6(2):133–144, 2012.
[19] Neo Technology. The Neo4j Manual v2.0.0, 2013.
[20] O. Pivert, O. Slama, and V. Thion. SPARQL extensions with preferences: a survey. In *Proceedings of th ACM Symposium on Applied Computing (SAC)*, pages 1015–1020, 2016.
[21] O. Pivert, G. Smits, and V. Thion. Expression and Efficient Processing of Fuzzy Queries in a Graph Database Context. In *Proceedings of IEEE International Conference on Fuzzy Systems (Fuzz-IEEE)*, page 8, Aug. 2015.
[22] RabbitHole. neo4j.com/blog/rabbithole-the-neo4j-repl-console/.
[23] K. Stefanidis, G. Koutrika, and E. Pitoura. A survey on representation, composition and application of preferences in database systems. *ACM Transactions on Database Systems (TODS)*, 36(3):19:1–19:45, Aug. 2011.
[24] M. Y. Vardi. On the complexity of bounded-variable queries. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 266–276, 1995.
[25] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins. A comparison of a graph database and a relational database: a data provenance perspective. In *Proceedings of the ACM Southeast Regional Conference*, page 42, 2010.
[26] A. Zaveri, D. Kontokostas, M. A. Sherif, L. Bühmann, M. Morsey, S. Auer, and J. Lehmann. User-driven quality evaluation of DBpedia. In *Proceedings of the Intl. Conf. on Semantic Systems (I-SEMANTICS)*, pages 97–104, 2013.
[27] A. Zaveri, A. Rula, A. Maurino, R. Pietrobon, J. Lehmann, and S. Auer. Quality assessment for linked data: A survey. *Semantic Web*, 7(1):63–93, 2016.