Fuzzy Quantified Structural Queries to Fuzzy Graph Databases

Olivier Pivert(✉), Olfa Slama, and Virginie Thion

University of Rennes 1 – Irisa, Lannion, France
{olivier.pivert,olfa.slama,virginie.thion}@irisa.fr

Abstract. This paper deals with *fuzzy quantified queries* in a graph database context. We study a particular type of structural quantified query and show how it can be expressed in the language FUDGE that we previously proposed. A processing strategy based on a compilation mechanism that derives regular (nonfuzzy) queries for accessing the relevant data is also described.

Keywords: Graph databases · Fuzzy quantified queries

1 Introduction

Even though the concept of a graph database is not exactly new [2], it is only recently that the database community has started to show a strong interest in it, due in particular to the rise of linked data on the Web and the profusion of domains where networked objects have to be handled: social networks, genomics, cartographic databases, etc.

Simultaneously, the need for flexible querying has been acknowledged by database researchers, and many approaches to relational database preference queries have been proposed in the last decade, see e.g. [14]. However, the pioneering work in this domain dates back to the 70's and is based on fuzzy set theory [15]. Since then, much effort has been made to come up with expressive and efficient flexible querying tools based on fuzzy logic, see e.g. [9]. In particular, *fuzzy quantified queries* have proved useful in a relational database context for expressing different types of imprecise information needs [4]. In a graph database context, such queries have an even higher potential since they can exploit the *structure* of the graph, beside the attribute values attached to the nodes or edges. Nevertheless, only one approach from the literature, described in [5], considered *fuzzy quantified queries* so far, and only in a limited way. In the present paper, we intend to integrate *fuzzy quantified queries* in a framework that we defined previously in [10,11].

The remainder of the paper is organized as follows. Section 2 presents the different elements that constitute the context of the work. Section 3 is a refresher about fuzzy quantified statements. Section 4 discusses related work. In Sect. 5, we consider a specific type of fuzzy quantified structural query, we propose a syntactic format for expressing it in the FUDGE language defined in [10], and

S. Schockaert and P. Senellart (Eds.): SUM 2016, LNAI 9858, pp. 260–273, 2016.
DOI: 10.1007/978-3-319-45856-4_18

we describe its interpretation. Section 6 deals with query processing. Finally, Sect. 7 recalls the main contributions and outlines research perspectives.

2 Background Notions

In this section, we recall important notions about graph databases, fuzzy graph theory, fuzzy graph databases, and the query language FUDGE.

2.1 Graph Databases

A graph database management system enables managing data for which the structure of 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. Different models of graph databases have been proposed in the literature (see [2] for an overview), including the *attributed graph* (aka. *property graph*) aimed to model a network of entities with embedded data. In this model, nodes and edges may contain data in *attributes* (aka. *properties*).

2.2 Fuzzy Graphs

A *graph* G is a pair (V, R), where V is a set and R is a relation on V. The elements of V (resp. R) correspond to the vertices (resp. edges) of the graph. Similarly, any fuzzy relation ρ on a set V can be regarded as defining a weighted graph, or fuzzy graph [13], where the edge $(x, y) \in V \times V$ has weight or strength $\rho(x, y) \in [0, 1]$.

An important operation on fuzzy relations is composition. Assume ρ_1 and ρ_2 are two fuzzy relations on V. Thus, composition $\rho = \rho_1 \circ \rho_2$ is also a fuzzy relation on V s.t. $\rho(x, z) = \max_y \min(\rho_1(x, y), \rho_2(y, z))$. The composition operation can be shown to be associative: $(\rho_1 \circ \rho_2) \circ \rho_3 = \rho_1 \circ (\rho_2 \circ \rho_3)$. The associativity property allows us to use the notation $\rho^k = \rho \circ \rho \circ \ldots \circ \rho$ for the composition of ρ with itself $k - 1$ times. In addition, following [16], we define ρ^0 to be s. t. $\rho^0(x, y) = 0$, $\forall(x, y)$.

Useful notions related to fuzzy graphs are those of strength and length of a path. Their definition, drawn from [13], is given hereafter.

Strength of a path. — A path p in G is a sequence $x_0 \rightarrow x_1 \rightarrow \ldots \rightarrow x_n$ $(n \geq 0)$ such that $\rho(x_{i-1}, x_i) > 0$, $1 \leq i \leq n$ and where n is the number of links in the path. The *strength* of the path is defined as

$$ST(p) = \min_{i=1..n} \rho(x_{i-1}, x_i). \tag{1}$$

In other words, the strength of a path is defined to be the weight of the weakest edge of the path. Two nodes for which there exists a path p with $ST(p) > 0$ between them are called *connected*. We call p a cycle if $n \geq 2$ and $x_0 = x_n$. It is possible to show that $\rho^k(x, y)$ is the strength of the strongest path from x to y

containing at most k links. Thus, the strength of the strongest path joining any two vertices x and y (using any number of links) may be denoted by $\rho^{\infty}(x, y)$.

Length and *distance*. — The *length* of a path $p = x_0 \rightarrow x_1 \rightarrow \ldots \rightarrow x_n$ in the sense of ρ is defined as follows:

$$Length(p) = \sum_{i=1}^{n} \frac{1}{\rho(x_{i-1}, x_i)}. \tag{2}$$

Clearly $Length(p) \geq n$ (it is equal to n if ρ is Boolean, i.e., if G is a nonfuzzy graph). We can then define the *distance* between two nodes x and y in G as

$$Distance(x, y) = \min_{all\ paths\ x\ to\ y} Length(p). \tag{3}$$

It is the length of the shortest path from x to y. It can be shown that *Distance* is a metric [13], i.e., $Distance(x, x) = 0$, $Distance(x, y) = Distance(y, x)$, and $Distance(x, z) \leq Distance(x, y) + Distance(y, z)\ \forall z$.

2.3 Fuzzy Graph Databases

We are interested in fuzzy graph databases where nodes and edges can carry data (e.g. key-value pairs in attributed graphs). So, we consider an extension of the notion of a *fuzzy graph*: the *fuzzy data graph* as defined in [11].

Definition 1 (Fuzzy data graph). *Let E be a set of labels. A* fuzzy data graph *G is a quadruple (V, R, κ, ζ), where V is a finite set of nodes (each node n is identified by $n.id$), $R = \bigcup_{e \in E}\{\rho_e : V \times V \rightarrow [0, 1]\}$ is a set of labeled fuzzy edges between nodes of V, and κ (resp. ζ) is a function assigning a (possibly structured) value to nodes (resp. edges) of G.*

In the following, a *graph database* is meant to be a fuzzy data graph. Figure 1 is an example of a fuzzy data graph in which the degree associated with `A -contributor-> B` is the proportion of journal papers co-written by `A` and `B`, over the total number of journal papers written by `B`. The degree associated with `J -domain-> D` is the extent to which the journal `J` belongs to the research domain `D`.

Nodes are assumed to be typed. If n is a node of V, then $Type(n)$ denotes its type. In Fig. 1, the nodes `IJWS12`, `IJAR14`, `IJIS16`, `IJIS10` and `IJUFK15` are of type *journal*, the nodes `IJWS12-p`, `IJAR14-p`, `IJIS16-p`, `IJIS10-p`, `IJIS10-p1` and `IJUFK15-p` of type *paper*, and the nodes `Andreas`, `Peter`, `Maria`, `Claudio`, `Michel`, `Bazil` and `Susan` are of type *author*, the nodes named `Database` are of type *domain* and the other nodes are of type *impact_factor*. For nodes of type *journal*, *paper*, *author* and *domain*, a property, called *name*, contains the identifier of the node and for nodes of type *impact_factor*, a property, called *value*, contains the value of the node. In Fig. 1, the value of the property *name* or *value* for a node appears inside the node.

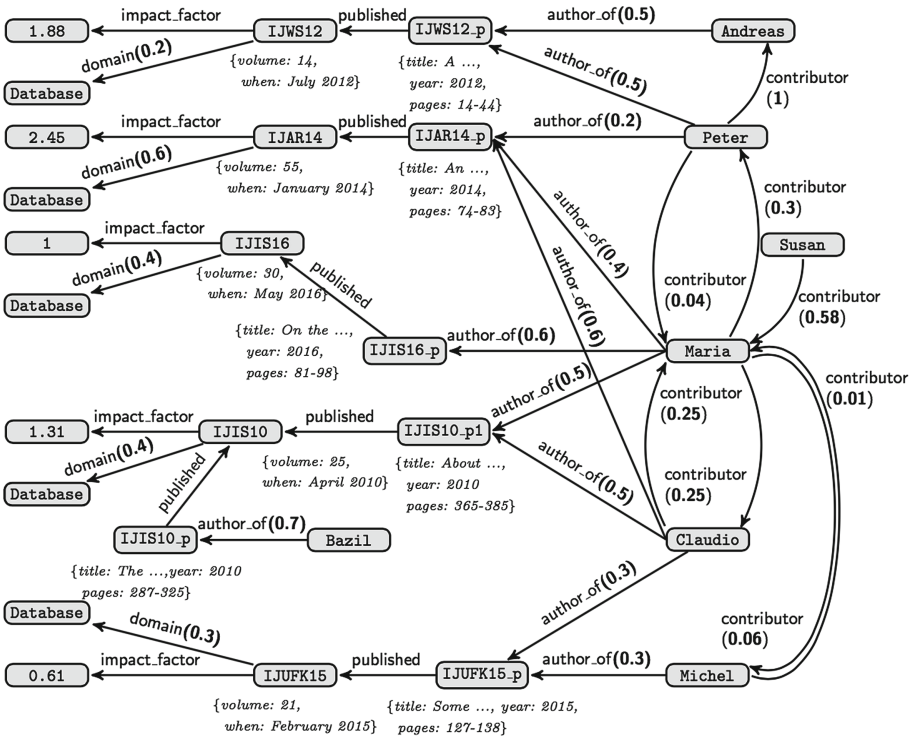


Fig. 1. Fuzzy data graph $\mathcal{DB}$

2.4 The FUDGE Query Language

FUDGE, based on the algebra described in [10], is an extension of the Cypher language [8], used in the Neo4j graph DBMS [1]. These languages are based on graph pattern matching, meaning that a query Q over a fuzzy data graph $\mathcal{DB}$ defines a graph pattern and answers to Q are its isomorphic subgraphs that can be found in $\mathcal{DB}$. More concretely, a pattern has the form of a subgraph where variables can occur. An answer maps the variables in elements of $\mathcal{DB}$.

A fuzzy graph pattern expressed *à la Cypher* consists of a set of expressions `(n1:Type1)-[exp]->(n2:Type2)` or `(n1:Type1)-[e:label]->(n2:Type2)` where `n1` and `n2` are node variables, `e` is an edge variable, `label` is a label of E, `exp` is a fuzzy regular expression, and `Type1` and `Type2` are node types. Such an expression denotes a path satisfying a fuzzy regular expression `exp` (that is *simple* in the second form `e`) going from a node of type `Type1` to a node of type `Type2`. All its arguments are optional, so the simplest form of an expression is `()-[]->()` denoting a path made of two nodes connected by any edge. Conditions on attributes are expressed on nodes and edges variables in a `WHERE` clause.

Example 1. We denote by $\mathcal{P}$ the fuzzy graph pattern:

```
1 MATCH
2   (au2)-[:contributor+]->(au1:author),
3   (au1)-[:author_of]->(ar1:paper), (ar1)-[:published]->(j1),
4   (au1)-[:author_of]->(ar2:paper), (ar2)-[:published]->(j2)
5 WHERE j1.name="IJWS12"
```

Listing 1.1. Pattern expressed *à la Cypher*

This pattern "models" information concerning authors (`au2`) who have, among their close contributors, an author (`au1`) who published a paper (`ar1`) in `IJWS12` and also published a paper (`ar2`) in a journal (`j2`). ◇

Let us illustrate the way a selection query can be expressed in FUDGE, that embarks fuzzy preferences over the data and the structure specified in the graph pattern. Given a graph database $\mathcal{DB}$, a selection query expressed in FUDGE is composed of:

1. A list of `DEFINE` clauses for fuzzy term declarations. If a fuzzy term `fterm` corresponds to a trapezoidal function defined by the quadruple (`A-a`, `A`, `B` and `B+b`), then the clause has the form `DEFINE fterm AS (A-a,A,B,B+b)`. If `fterm` is a decreasing function, then the clause has the form `DEFINEDESC fterm AS` (δ,γ) meaning that the support of the term is $[0, \gamma]$ and its core $[0, \delta]$ (there is the corresponding `DEFINEASC` clause for increasing functions).
2. A `MATCH` clause, which has the form `MATCH pattern WHERE conditions` that expresses the fuzzy graph pattern.

Example 2. Listing 1.2 is an example of a FUDGE query.

```
1 DEFINEDESC short AS (3,5), DEFINEASC high AS (0.5,2) IN
2 MATCH
3   (au2)-[(contributor+)|Length IS short]->(au1:author),
4   (au1)-[:author_of]->(ar1:paper), (ar1)-[:published]->(j1),
5   (au1)-[:author_of]->(ar2:paper), (ar2)-[:published]->(j2),
6   (j2)-[:impact_factor]->(i)
7 WHERE j1.name="IJWS12" AND i.value IS high
```

Listing 1.2. A FUDGE query

This pattern "models" information concerning authors (`au2`) who have, among their close contributors (connected by a short path — `Length IS short` — made of `contributor` edges), an author (`au1`) who published a paper (`ar1`) in `IJWS12` and also published a paper (`ar2`) in a journal (`j2`) which has a *high* impact factor (`i.value IS high`). The fuzzy terms *short* and *high* are defined on line 1. ◇

3 Refresher on Fuzzy Quantified Statements

In this section, we recall important notions about fuzzy quantifiers and present one of the approaches that have been proposed in the literature for interpreting fuzzy quantified statements.

3.1 Fuzzy Quantifiers

Zadeh [17] distinguishes between absolute and relative fuzzy quantifiers. Absolute quantifiers refer to a number while relative ones refer to a proportion. Quantifiers may also be increasing, as "at least half", or decreasing, as "at most three". An absolute quantifier $\mathcal{Q}$ is represented by a function $\mu_{\mathcal{Q}}$ from an integer range to [0, 1] whereas a relative quantifier is a mapping $\mu_{\mathcal{Q}}$ from [0, 1] to [0, 1]. In both cases, the value $\mu_{\mathcal{Q}}(j)$ is defined as the truth value of the statement "$\mathcal{Q}\,X$ are A" when exactly j elements from X fully satisfy A (whereas it is assumed that A is fully unsatisfied for the other elements). Figure 2 gives two examples of monotonous decreasing and increasing quantifiers respectively.

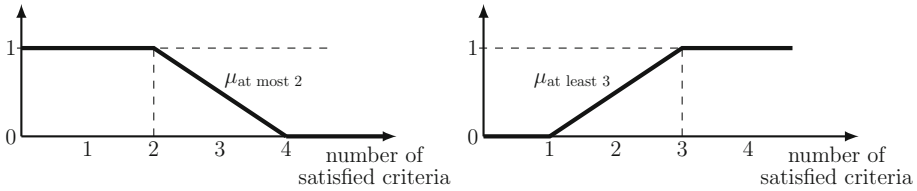


Fig. 2. Quantifiers "at most 2" (left) and "at least 3" (right)

Calculating the truth degree of the statement "$\mathcal{Q}X$ are A" raises the problem of determining the cardinality of the set of elements from X which satisfy A. If A is a Boolean predicate, this cardinality is a precise integer (k), and then, the truth value of "$\mathcal{Q}\,X$ are A" is $\mu_{\mathcal{Q}}(k)$. If A is a fuzzy predicate, this cardinality cannot be established precisely and then, computing the quantification corresponds to establishing the value of function $\mu_{\mathcal{Q}}$ for an imprecise argument.

3.2 Zadeh's Interpretation

Let X be the usual (crisp) set $\{x_1, x_2, \ldots, x_n\}$. Zadeh [17] defines the cardinality of the set of elements of X which satisfy A, denoted by $\Sigma count(A)$, as:

$$\Sigma count(A) = \sum_{i=1}^{n} \mu_A(x_i) \tag{4}$$

The truth degree of the statement "$\mathcal{Q}\,X$ are A" is then given by

$$\mu(\mathcal{Q}\,X \textit{ are } A) = \begin{cases} \mu_{\mathcal{Q}}(\Sigma count(A)) & \text{(absolute)}, \\ \mu_{\mathcal{Q}}\left(\dfrac{\Sigma count(A)}{n}\right) & \text{(relative)} \end{cases} \tag{5}$$

where n denotes the cardinality of X.

As for quantified statements of the form "$\mathcal{Q}\ B\ X$ are A" (with $\mathcal{Q}$ relative), their interpretation is as follows:

$$\mu(\mathcal{Q}BXareA) = \mu_{\mathcal{Q}}\left(\frac{\Sigma count(A \cap B)}{\Sigma count(B)}\right) = \mu_{\mathcal{Q}}\left(\frac{\sum_{x \in X} \top(\mu_A(x),\ \mu_B(x))}{\sum_{x \in X} \mu_B(x)}\right) \quad (6)$$

where $\top$ denotes a triangular norm (for instance the minimum).

4 Related Work

Fuzzy quantified queries have been thoroughly studied in a relational database context, see e.g. [4,7] where they serve to express conditions about data *values*. In a graph database context, a new dimension can be exploited that concerns the *structure* of the graph. In [16], Yager briefly mentions the possibility of using *fuzzy quantified queries* in a social network database context, such as the question of whether "*most* of the people residing in *western* countries have *strong* connections with each other" and suggests to interpret it using an OWA operator. However, the author does not propose any formal language for expressing such queries.

A first attempt to extend Cypher with *fuzzy quantified queries* — in the context of a *regular* (crisp) graph database — is described in [5,6]. In [5], the authors take as an example a graph database representing hotels and their customers and consider the following fuzzy quantified query:

```
MATCH (c1:customer)-[:knows**almost3]->(c2:customer) RETURN c1,c2
```

looking for pairs of customers linked through *almost 3* hops. The syntax `**` is used for indicating what the authors call a *fuzzy linker*. However, the interpretation of such queries is not formally given. The authors give a second example that involves the fuzzy concept *popular* applied to hotels. They assume that a hotel is popular if a large proportion of customers visited it. First, they consider a crisp interpretation of this concept (*large* being seen as equivalent to *at least* n) and recall how the corresponding query can be expressed in Cypher:

```
MATCH (c:customer)-[:visit]->(h:hotel) WITH h, count(*) AS cpt
WHERE cpt > n − 1 RETURN h
```

Then, the authors switch to a fuzzy interpretation of the term *popular* and propose the expression:

```
MATCH (c:customer)-[:visit]->(h:hotel) WITH h, count(*) AS cpt
WHERE popular(cpt) > 0 RETURN h
```

In [6], the same authors propose an approach aimed to summarize a (crisp) graph database by means of fuzzy quantified statements of the form $\mathcal{Q}\ X$ *are* A, in the same spirit as what Rasmussen and Yager did for relational databases [12]. Again, they consider that the degree of truth of such a statement is obtained by a sigma-count (according to Zadeh's interpretation) and show how the corresponding queries can be expressed in Cypher. More precisely, given a graph database

G and a summary $S = a$–$[r]$–>b, $\mathcal{Q}$, the authors consider two degrees of truth of S in G defined by $truth_1(S) = \mu_{\mathcal{Q}}\,(count(distinct\ S)/count(distinct\ a))$ and $truth_2(S) = \mu_{\mathcal{Q}}\,(count(distinct\ S)/count(distinct\ a$–$[r]$–>$(?)))$. They illustrate these notions using a database representing students who rent or own a house or an apartment. The degree of truth (in the sense of the second formula above) of the summary "S = *student*–[*rent*]–>*apartment*, *most*" — meaning "most of the students rent an apartment" (as opposed to a house) — is given by the membership degree to the fuzzy quantifier *most* of the ratio: (number of times a relationship of type *rents* appears between a student and an apartment) over (number of relations of type *rents* starting from a *student* node).

A limitation of this approach is that only the quantifier is fuzzy (whereas in general, in a fuzzy quantified statement of the form "$\mathcal{Q}$ B X are A", the predicates A and B may be fuzzy too).

5 Fuzzy Quantified Queries in the FUDGE Language

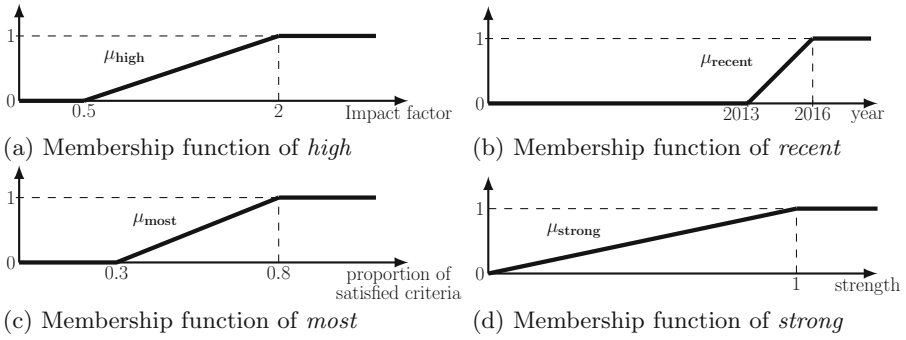
In the following, we consider *fuzzy quantified queries* involving fuzzy predicates (beside the quantifier) over *fuzzy* graph databases. The fuzzy quantified statements considered are of the form "$\mathcal{Q}$ nodes, that are connected according to a certain pattern to a node x, satisfy a fuzzy condition φ". An example of such a statement is: "*most* of the papers of which x is a *main* author, have been published in a *renowned* database journal".

This type of statement rewrites "$\mathcal{Q}$ $Y_{P(x)}$ are φ" where the quantifier $\mathcal{Q}$ is represented by a fuzzy set and denotes either a relative quantifier (e.g., most) or an absolute one (e.g., at least three), $Y_{P(x)}$ designates the fuzzy set of nodes connected, according to the pattern $P(x)$, to a node x in the graph, and φ, is represented also by a fuzzy set and denotes fuzzy (possibly compound) conditions. In a general setting, we have a statement of the form "$\mathcal{Q}$ B X are A" where B is the fuzzy condition "to be connected (according to the pattern $P(x)$) to a node x", X is the set of nodes in the graph, and A is the fuzzy condition φ. In the particular case where the graph is crisp, we get a statement of the form "$\mathcal{Q}$ X are A" where the referential X is the (crisp) set of nodes connected to x.

Example 3. The query that consists in finding "*most* of the papers of which x is a *main* author, have been published in a *renowned* database journal" may be expressed in FUDGE as follows:

```
1  DEFINEQRELATIVEASC most AS (0.3,0.8),
2  DEFINEASC strong AS (0,1), DEFINEASC high AS (0.5,2) IN
3  MATCH
4    (x:author)-[author_of|ST IS strong]->(p:paper),
5    (p:paper)-[:published]->(j:journal)-[:impact_factor]->(i:impact_factor),
6    (j:journal)-[:domain]->(d)
7  WITH x HAVING most(p) ARE (i.value IS high AND d.name="database")
8  RETURN x
```

where the DEFINEQRELATIVEASC clause defines the fuzzy relative increasing quantifier *most* of Fig. 3(c), and the next DEFINEASC clauses define the ascending fuzzy terms *strong* and *high* of Fig. 3(d) and (a).



(a) Membership function of *high*

(b) Membership function of *recent*

(c) Membership function of *most*

(d) Membership function of *strong*

Fig. 3. Membership functions

We now consider a slightly more complex version of the above example by adding a fuzzy condition on the papers' publication date: "*most* of the recent papers written by an important author x have been published in a *renowned* database journal". The syntactic form of this query, denoted by $Q_{mostAuthors}$ in the following, is given in Listing 1.3. ◇

```
1 DEFINEQRELATIVEASC most AS (0.3,0.8), DEFINEASC recent AS (2013,2016),
2 DEFINEASC strong AS (0,1), DEFINEASC high AS (0.5,2) IN
3 MATCH
4   (x:author)-[author_of|ST IS strong]->(p:paper),
5   (p:paper)-[:published]->(j:journal)-[:impact_factor]->(i:impact_factor),
6   (j:journal)-[:domain]->(d)
7 WHERE p.year IS recent
8 WITH x HAVING most(p) ARE (i.value IS high AND d.name="database")
9 RETURN x
```

Listing 1.3. Syntax of the fuzzy quantified query $Q_{mostAuthors}$

The general syntactic form of *fuzzy quantified queries* is given in the Listing 1.4.

```
1 DEFINE... IN
2 MATCH P(x,y) WHERE fc1(y)
3 WITH x HAVING Quant(y) ARE fc2
4 RETURN x
```

Listing 1.4. Syntax of a fuzzy quantified query

It contains a list of DEFINE clauses for the fuzzy quantifiers and the fuzzy terms declarations, a MATCH clause for fuzzy graph pattern selection, a WHERE clause for expressing the (possibly fuzzy) conditions on values, a HAVING clause for the fuzzy quantified statement definition, and a RETURN clause for specifying which elements should be returned in the resultset. P(x,y) denotes the fuzzy graph pattern involving the nodes x and y. fc_1 and fc_2 are fuzzy conditions.

Interpretation: From a conceptual point of view, its interpretation involves three derived queries (hereafter, the DEFINE clauses are been omitted for the sake of simplicity). The first one, Q_1 (given in Listing 1.5), aims to retrieve the elements matching the variable x, for which we will then need to calculate a satisfaction degree. Query Q_1 is obtained by removing the WITH and HAVING clauses from the initial query (one may also remove some useless parts of P(x,y), as illustrated in Example 4 below).

```
1 MATCH P(x,y) WHERE fc1(y)
2 RETURN x
```

Listing 1.5. Derived query Q_1

The second derived query, denoted by $Q_2(e)$ (given in Listing 1.6), where e is an element returned by Q_1, is obtained by removing the WITH and HAVING clauses from the initial query, integrating the fuzzy condition fc_2 and the condition x.name=e in the WHERE clause and adding the clause RETURN y. According to the semantics of a FUDGE query, its result, denoted by $A_{Q_2(x)}$, is a set of elements $\{(\mu_1/y_1), ..., (\mu_n/y_n)\}$, where μ_i is the satisfaction degree associated with the element y_i.

```
1 MATCH P(x,y) WHERE fc1(y) AND fc2 AND x.name=e
2 RETURN y
```

Listing 1.6. Derived queries $Q_2(e)$ for each e retrieved by Q_1

The third derived query, denoted by $Q_3(e)$ (given in Listing 1.7), is the initial fuzzy query from the MATCH to the WHERE clause, adding the condition x.name=e in the WHERE clause and the clause RETURN y as follows:

```
1 MATCH P(x,y) WHERE fc1(y) AND x.name=e
2 RETURN y
```

Listing 1.7. Derived queries $Q_3(e)$ for each e retrieved by Q_1

The result of this query, denoted by $A_{Q_3(x)}$, takes the form of a set of elements $\{(\mu'_1/y_1), ..., (\mu'_m/y_m)\}$, where μ'_i is the satisfaction degree associated with the element y_i. Note that Q_3 only differs from Q_2 by its WHERE clause.

In accordance with the semantics of the projection, if the same value of y_i appears in several instances in the resultset of $Q_2(x)$ or $Q_3(x)$, duplicates are eliminated and the final degree associated with y_i in $A_{Q_2(x)}$ and $A_{Q_3(x)}$ is equal to the maximum degree associated with these occurrences.

Then, the results of the initial fuzzy relative quantified query Q (involving the fuzzy quantifier $\mathcal{Q}$) are results of the query Q_1 derived from Q, and the final satisfaction degree associated with each element e of these results is

$$\mu(e) = \mu_{\mathcal{Q}} \left(\frac{\sum_{(\mu_i/y_i) \in A_{Q_2(e)}} \mu_i}{\sum_{(\mu'_i/y_i) \in A_{Q_3(e)}} \mu'_i} \right) \tag{7}$$

In case of a fuzzy absolute quantified query, the final satisfaction degree associated with each element e is $\mu(e) = \mu_{\mathcal{Q}}\left(\sum_{(\mu_i/y_i)\in A_{Q_2(e)}} \mu_i\right)$.

Example 4. Let us consider the query $Q_{mostAuthors}$ of Listing 1.3. We evaluate this query according to the fuzzy data graph $\mathcal{DB}$ of Fig. 1. In order to interpret $Q_{mostAuthors}$, we first evaluate the following query Q_1, derived from $Q_{mostAuthors}$, that retrieves "the authors (x) who highly contributed to at least one recent paper (p) published in a journal".

```
1 MATCH (x:author)-[author_of|ST IS strong]->(p:paper),
2  (p:paper)-[:published]->(j:journal)
3 WHERE p.year IS recent
4 RETURN x
```

Listing 1.8. Query Q_1 derived from $Q_{mostAuthors}$

Q_1 returns four results X = {Peter, Maria, Claudio, Michel}. The authors Andreas, Susan and Bazil do not belong to the resultset of Q_1 because Susan has not written a journal paper yet and Andreas and Bazil do not have a recent paper.

For each element x from the resultset X of Q_1, we process two queries $Q_2(x)$ and $Q_3(x)$. The query $Q_2(x)$, derived from $Q_{mostAuthors}$, aims to retrieve "the recent papers of which x is a *main* author, that have been published in a *renowned* database journal". For instance, for the element Maria, the query $Q_2(Maria)$ is expressed as follows:

```
1 MATCH (x:author)-[author_of|ST IS strong]->(p:paper),
2  (p:paper)-[:published]->(j:journal)-[:impact_factor]->(i:impact_factor),
3  (j:journal)-[:domain]->(d)
4 WHERE p.year IS recent AND i.value IS high
5   AND d.name="database" AND x.name="Maria"
6 RETURN p
```

Listing 1.9. Query $Q_2(Maria)$ derived from $Q_{mostAuthors}$

For a given x, we get a list of papers with their respective satisfaction degrees: $\mu(p) = \min(\mu_{strong}(\rho_{author}(x,p)), \mu_{recent}(p), \mu_{high}(i))$. For the running example, we then have $A_{Q_2(Peter)}$ = {(min(0.5, 0, 0.92)/IJWS12_p), (min(0.2, 0.33, 1)/IJAR14_p)} = {(0/IJWS12_p), (0.2/IJAR14_p)}, $A_{Q_2(Maria)}$ = {(0.33/IJAR14_p), (0.33/IJIS16_p), (0/$IJIS10_p$1)}, $A_{Q_2(Claudio)}$ = {(0.33/IJAR14_p), (0/IJIS10_p1), (0.07/IJUFK15_p)}, $A_{Q_2(Michel)}$ = {(0.07/IJUFK15_p)}.

Query $Q_3(x)$, derived from $Q_{mostAuthors}$, aims to retrieve "the recent papers of which x is a *main* author, that have been published in a journal". For instance, for the element Maria, the query $Q_3(Maria)$ is expressed as follows:

```
1 MATCH (x:author)-[author_of|ST IS strong]->(p:paper),
2  (p:paper)-[:published]->(j:journal)-[:impact_factor]->(i:impact_factor),
3  (j:journal)-[:domain]->(d)
4 WHERE p.year IS recent AND x.name="Maria"
5 RETURN p
```

Listing 1.10. Query $Q_3(Maria)$ derived from $Q_{mostAuthors}$

For a given x, we get a set of papers written by x satisfying the conditions of query $Q_3(x)$ with their respective satisfaction degrees as follows: $\mu(p) = min(\mu_{strong}(\rho_{author}(x,p)), \mu_{recent}(p))$. For the running example, we then have $A_{Q_3(Peter)} = \{(0/\texttt{IJWS12\_p}), (0.2/\texttt{IJAR14\_p})\}$, $A_{Q_3(Maria)} = \{(0.33/\texttt{IJAR14\_p}), (0.6/\texttt{IJIS16\_p}), (0/\texttt{IJIS10\_p1})\}$, $A_{Q_3(Claudio)} = \{(0.33/\texttt{IJAR14\_p}), (0/\texttt{IJIS10\_p1}), (0.3/\texttt{IJUFK15\_p})\}$, $A_{Q_3(Michel)} = \{(0.3/\texttt{IJUFK15\_p})\}$.

Lastly, the final result of the query $Q_{mostAuthors}$ evaluated on $\mathcal{DB}$, given by Formula 7, is $\{\mu(Peter) = \mu_{most}(\frac{0.2}{0.2}) = 1, \mu(Maria) = \mu_{most}(\frac{0.66}{0.93}) = 0.82, \mu(Claudio) = \mu_{most}(\frac{0.4}{0.63}) = 0.67, \mu(Michel) = \mu_{most}(\frac{0.07}{0.3}) = 0\}$. ◇

6 About Query Processing

The evaluation strategy we propose for these queries consists of a software add-on layer over the *Neo4j* graph DBMS. This software, called SUGAR, efficiently evaluates FUDGE queries that contain fuzzy preferences, but its initial version, described in [10,11], does not support fuzzy quantified statements. We now consider the implementation of this functionality, based on the theoretical foundations defined in the previous section. The SUGAR software implements two modules, which interact with the embedded Neo4j crisp engine (see Fig. 4): *The Transcriptor module*, aimed to translate a FUDGE query requested by a user into a (crisp) cypher one (using the derivation principle presented in [9] in the context of relational databases), which is then sent to the crisp Neo4j engine, and *The Score Calculator module*, which calculates the satisfaction degree associated with each answer returned by the crisp engine, and ranks the answers.

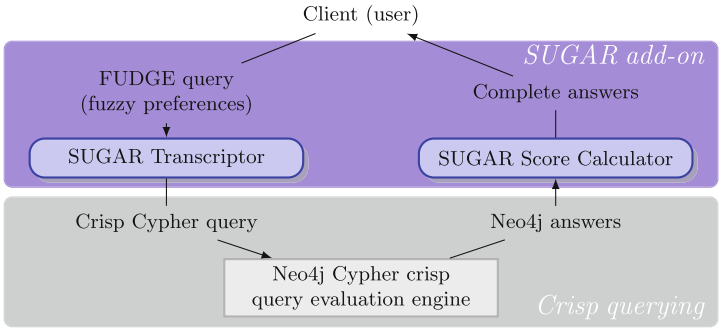


Fig. 4. SUGAR software architecture

The main process in our work is the quantified statement evaluation step which is described in Algorithm 1. For quantified queries of the type introduced in the previous section (i.e. using relative quantifiers), the principle is to first evaluate the fuzzy query Q_1. For each tuple x from the resulset of Q_1, we evaluate with SUGAR the fuzzy queries $Q_2(x)$ and $Q_3(x)$. The final satisfaction degree is given by Formula 7 according to $Q_2(x)$ and $Q_3(x)$ resultsets. Finally, we get as an output answers ranked by decreasing order of the satisfaction degree.

Algorithm 1. Algorithm for the evaluation of a fuzzy quantified query

Input : A query Q containing a fuzzy quantifier *Quant*
Output: Results X of Q with associated satisfaction degrees $\{\mu(x)|x \in X\}$

```
1 begin
2   Derive queries Q1, Q2 and Q3 from Q;
3   X=evaluate(Q1);
4   foreach element x from the result of X do
5     evaluate(Q2(x));
6     evaluate(Q3(x));
7     μ(x) = μ_Quant(μ_{A_{Q2(x)}}(y)/μ_{A_{Q3(x)}}(y));
8   Rank answers of X by decreasing satisfaction degree (μ)
```

For a given x, queries $Q_2(x)$ and $Q_3(e)$ embed the same graph pattern (they only differ by their `WHERE` clause that is more restrictive for Q_2). This means that these queries could be processed together at evaluation time. Then one can see on Algorithm 1 that evaluating a fuzzy quantified query implies processing $x+1$ FUDGE queries where x is the number of elements that match the pattern declared in the `MATCH` clause of the initial query (without the quantified statement). The cost of the evaluation of a graph pattern query depends on the form of its pattern [3] and it has already been showed in [10] that a FUDGE query does not significantly increase the cost with respect to a crisp query in the case of selection graph pattern queries.

As a proof-of-concept of the proposed approach, the FUDGE prototype is available and downloadable at http://www-shaman.irisa.fr/fudge-prototype/.

7 Conclusion and Perspectives

In this paper, we have dealt with a specific type of *fuzzy quantified queries*, addressed to fuzzy graph databases. We have defined the syntax and semantics of an extension of the query language Cypher that makes it possible to express and interpret such queries. A query processing strategy based on the derivation of non-quantified fuzzy queries has also been proposed. As a future work, we first intend to carry out some experimentations in order to assess the performances of the evaluation method outlined here. We then plan to study other types of fuzzy quantified queries. An example of a fuzzy quantified statement that is out of the scope of the present approach is "find the authors x that had a paper published in *most* of the *renowned* database journals". More generally, it would be interesting to study fuzzy quantified queries that aim to find the nodes x such that x is connected (by a path) to $\mathcal{Q}$ nodes reachable by a given pattern and satisfying a given condition C.

Acknowledgement. This work has been partially funded by the French DGE (Direction Générale des Entreprises) under the project ODIN.

References

1. Neo4j web site. www.neo4j.org
2. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Comput. Surv. **40**(1), 1–39 (2008)
3. Barceló, P., Libkin, L., Reutter, J.L.: Querying regular graph patterns. J. ACM **61**(1), 8:1–8:54 (2014)
4. Bosc, P., Liétard, L., Pivert, O.: Quantified statements and database fuzzy querying. In: Bosc, P., Kacprzyk, J. (eds.) Fuzziness in Database Management Systems, pp. 275–308. Physica Verlag, Heidelberg (1995)
5. Castelltort, A., Laurent, A.: Fuzzy queries over noSQL graph databases: perspectives for extending the cypher language. In: Laurent, A., Strauss, O., Bouchon-Meunier, B., Yager, R.R. (eds.) IPMU 2014, Part III. CCIS, vol. 444, pp. 384–395. Springer, Heidelberg (2014)
6. Castelltort, A., Laurent, A.: Extracting fuzzy summaries from NoSQL graph databases. In: Andreasen, T., et al. (eds.) FQAS'15. AISC, vol. 400, pp. 189–200. Springer, Switzerland (2015)
7. Kacprzyk, J., Zadrożny, S., Ziółkowski, A.: FQUERY III +: a "human-consistent" database querying system based on fuzzy logic with linguistic quantifiers. Inf. Syst. **14**(6), 443–453 (1989)
8. Neo Technology: The Neo4j Manual v2.0.0, part III (2013)
9. Pivert, O., Bosc, P.: Fuzzy Preference Queries to Relational Databases. Imperial College Press, London (2012)
10. Pivert, O., Smits, G., Thion, V.: Expression and efficient processing of fuzzy queries in a graph database context. In: Proceedings of the 24th IEEE International Conference on Fuzzy Systems (Fuzz-IEEE 2015), Istanbul, Turkey (2015)
11. Pivert, O., Thion, V., Jaudoin, H., Smits, G.: On a fuzzy algebra for querying graph databases. In: Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2014), pp. 748–755, Limassol, Cyprus (2014)
12. Rasmussen, D., Yager, R.R.: Summary SQL - a fuzzy tool for data mining. Intell. Data Anal. **1**(1–4), 49–58 (1997)
13. Rosenfeld, A.: Fuzzy graphs. In: Fuzzy Sets and their Applications to Cognitive and Decision Processes, pp. 77–97. Academic Press, London (1975)
14. Stefanidis, K., Koutrika, G., Pitoura, E.: A survey on representation, composition and application of preferences in database systems. ACM Trans. Database Syst. **36**(3), 19 (2011). http://doi.acm.org/10.1145/2000824.2000829
15. Tahani, V.: A conceptual framework for fuzzy query processing - a step toward very intelligent database systems. Inf. Process. Manag. **13**(5), 289–303 (1977)
16. Yager, R.R.: Social network database querying based on computing with words. In: Pivert, O., Zadrożny, S. (eds.) Flexible Approaches in Data, Information and Knowledge Management. SCI, vol. 497, pp. 241–257. Springer, Switzerland (2013)
17. Zadeh, L.A.: A computational approach to fuzzy quantifiers in natural languages. Computi. Math. Appl. **9**, 149–183 (1983)