Expression and efficient evaluation of fuzzy quantified structural queries to fuzzy graph databases

Olivier Pivert, Olfa Slama, Virginie Thion

Expression and Efficient Evaluation of Fuzzy Quantified Structural Queries to Fuzzy Graph Databases

Olivier Pivert, Olfa Slama, Virginie Thion

University of Rennes 1 – IRISA, Lannion, France
firstname.lastname@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 an extension of the Neo4j Cypher query language. A processing strategy based on a compilation mechanism that derives regular (nonfuzzy) queries for accessing the relevant data is also described. Then, some experiments are performed that show the tractability of this approach.

Keywords: Graph Databases, Fuzzy Quantified Queries
2010 MSC: 00-01, 99-00

1. Introduction

Even though the concept of a graph database is not exactly new, cf. [1], 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. [2]. However, the pioneering work in this domain dates back to the 70's and is based on fuzzy set theory [3]. Since then, much effort has been made to come up with expressive and efficient flexible querying tools based on fuzzy logic, see e.g. [4]. In particular, *fuzzy quantified queries* have proved useful in a relational database context for expressing different types of imprecise information needs, see

e.g. [5]. 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 [6], considered *fuzzy quantified queries* so far and only in a limited way.

The present paper is an extended version of our work in [7] in which we have integrated *fuzzy quantified queries* in a framework named FUDGE that we defined previously in [8]. In this paper, we describe in more detail the syntactic form of the queries considered as well as a processing strategy aimed to efficiently evaluate them. Moreover, we provide some experimental results that show the tractability of the approach.

The remainder of the paper is organized as follows. Section 2 presents some background notions about graph databases, fuzzy graph theory, fuzzy graph databases, and the principles underlying the FUDGE query language introduced in [9]. Section 3 is a refresher about fuzzy quantified statements. Section 4 discusses related work. In Section 5, we consider a specific type of fuzzy quantified structural query, we propose a syntactic format for expressing it in the FUDGE language and we describe its interpretation. Section 6 deals with query processing and discusses implementation issues. In Section 7, some experimental results showing the feasibility of the approach are presented. Finally, Section 8 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 [1] 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, see [10], where the edge $(x, y) \in V \times V$ has weight or strength $\rho(x, y) \in [0, 1]$. Having no edge between x and y is equivalent to $\rho(x, y) = 0$.

A fuzzy data graph may contain both fuzzy edges and crisp edges as a fuzzy edge with a degree of 0 or 1 can be considered as crisp. Along the same line, a crisp data graph is simply a special case of fuzzy data graph (where $\rho : V \times V \to \{0, 1\}$ is Boolean). We then only deal with fuzzy edges and data graph in the following.

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 [11], 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 definitions, drawn from [10], are given hereafter.

Strength of a path. — A path p in G is a sequence $x_0 \to x_1 \to \ldots \to x_n$ ($n > 0$) where the x_i's are nodes from V, 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 $p = x_0 \to x_1 \to \ldots \to x_n$ 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

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

It is the length of the shortest path from x to y.

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 [8].

Definition 1 (Fuzzy data graph). *Let E be a set of labels. A* fuzzy data graph *G is a quadruple $(V,\ R,\ \kappa,\ \zeta)$, where V is a finite set of nodes (each node n is identified by n.id) and $R = \bigcup_{e \in E} \{\rho_e :\ V \times V \to [0,\ 1]\}$ is a set of labeled fuzzy edges between nodes of V. Data may be attached to nodes (resp. edges), in the form of a set of key-value pairs for each node (resp. edge), modeled through the κ (resp. ζ) function.*

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 (denoted by $\rho_{domain}(J, D)$ in Definition 1) is the extent to which the journal J belongs to the research domain D.

The nodes are assumed to be typed. If n is a node of V, then $Type(n)$ denotes its type. In Figure 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
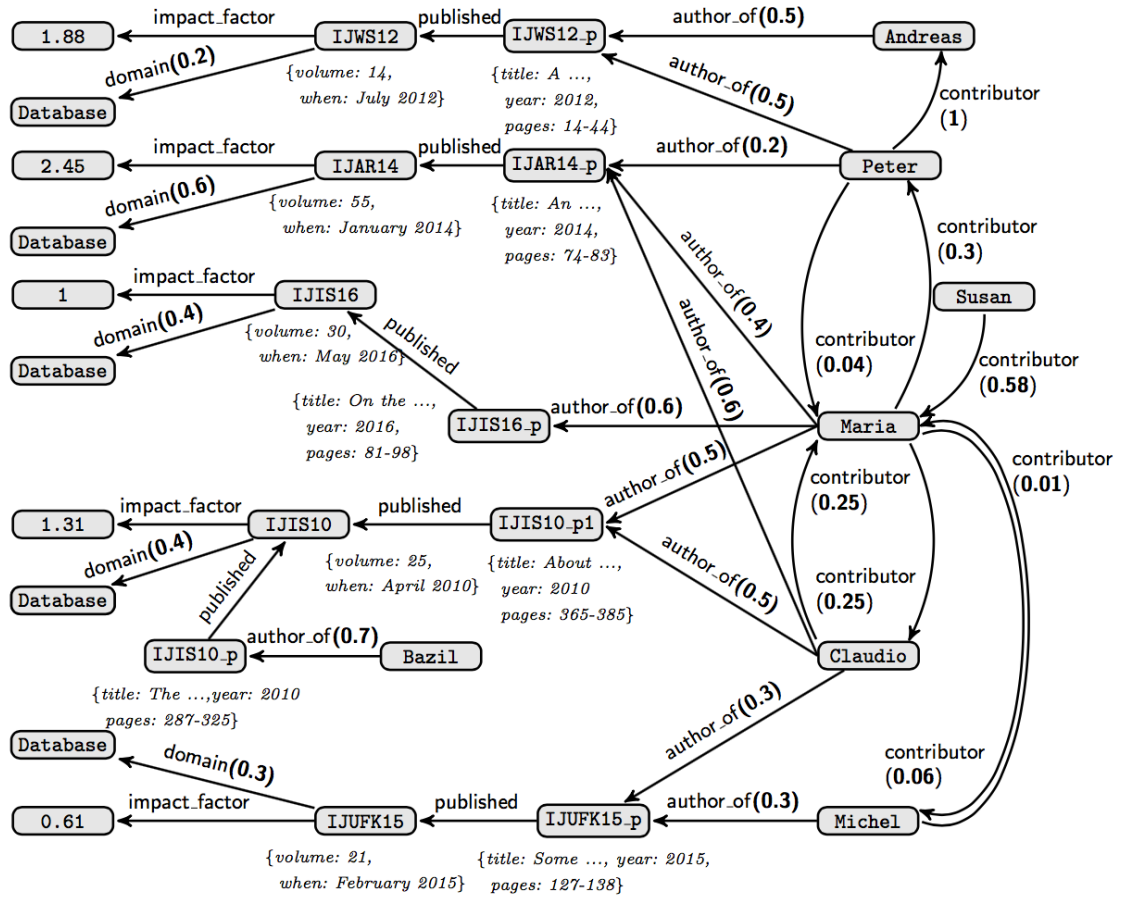
4

Figure 1: Fuzzy data graph $\mathcal{DB}$

impact_factor, a property, called *value*, contains the value of the node. In Figure 1, the value of the property *name* or *value* for a node appears inside the node. Data are attached to nodes only in this example. For instance here, $\kappa(\texttt{IJWS12}) = \{volume:\ 14,\ when:\ July\ 2012\}$.

₁₁₅ *2.4. The* FUDGE *Query Language*

FUDGE, based on the algebra described in [9], is an extension of the Cypher language (see [12]), initially used in the Neo4j graph DBMS (see [13]) and now in other systems too, thanks to the *OpenCypher* initiative. These languages are based on graph pattern matching, meaning that a query
₁₂₀ Q over a fuzzy data graph $\mathcal{DB}$ defines a graph pattern and answers to Q

5

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 graph pattern:*

```
1  MATCH
2    (au2:author)-[: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" AND j1.name <> j2.name
```
Listing 1: Pattern expressed *à la Cypher*

This pattern "models" information concerning authors (au2) such that there exists a related — in the sense of a `contributor` path — author (au1) who published a paper (ar1) in IJWS12 and also published a paper (ar2) in another journal (j2). ◇

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

1. A list of `DEFINE` clauses for fuzzy term declarations. We choose to use linear membership functions. If a fuzzy term `fterm` has a trapezoidal function defined by the quadruple (`A'`, `A`, `B`, `B'`) — meaning that its support is $]A', B'[$ and its core $[A, B]$ —, then the clause has the form `DEFINE fterm AS (A', A, 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 $[0, \delta]$. For an increasing function, the clause has the form `DEFINEASC fterm AS` (δ,γ) meaning that the support of the term is $]\delta, +\infty[$ and its core $[\gamma, +\infty[$.

6

2. A `MATCH` clause, which has the form `MATCH` pattern `WHERE` conditions that expresses the fuzzy graph pattern.

Example 2. *Listing 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:author)-[(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 j1.name <> j2.name AND i.value IS high
```
Listing 2: A FUDGE query

This pattern specifies information concerning authors (`au2`) such that there exists a closely related author — i.e., connected by a short path made of `contributor` edges — (`au1`) who published a paper (`ar1`) in `IJWS12` and also published a paper (`ar2`) in another 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, then, we present two approaches that have been proposed in the literature for interpreting fuzzy quantified statements.

3.1. Fuzzy Quantifiers

Fuzzy logic extends the notion of quantifier from Boolean logic and makes it possible to model quantifiers from the natural language (most of, at least half, few, around a dozen, etc).

[14] distinguishes between absolute and relative fuzzy quantifiers. Absolute quantifiers refer to a number while relative ones refer to a proportion.

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].

Quantifiers may also be increasing, as "at least half", or decreasing, as "at most three". According to [15], when an increasing (proportional) quantifier is used, if all of the x_i's fully satisfy A, then the statement "$\mathcal{Q}\,X$ are A" is entirely true and if all of the x_i's do not satisfy A at all, then the statement "$\mathcal{Q}\,X$ are A" is entirely false. Moreover, the transition between those two extremes is continuous and monotonous. Therefore, when $\mathcal{Q}$ is increasing (e.g.,
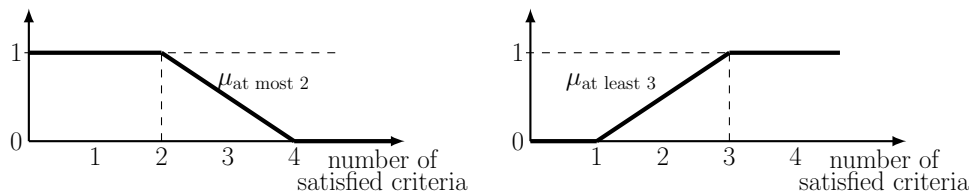
7

Figure 2: Quantifiers "at most 2" (left) and "at least 3" (right)

"most", "at least a half"), function μ_Q is increasing. Similarly, decreasing quantifiers (e.g., "at most two", "at most a half") are defined by decreasing functions.

Figure 2 gives two examples of monotonous decreasing and increasing fuzzy quantifiers respectively.

Calculating the truth degree of the statement "$Q\,X$ 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 "$Q\,X$ are A" is $\mu_Q(k)$. If A is a fuzzy predicate, this cardinality cannot be established precisely and then, computing the quantification corresponds to establishing the value of function μ_Q for an imprecise argument.

3.2. Interpretation of Fuzzy Quantified Statements

We now consider two proposals from the literature for interpreting quantified statements of the type "$Q\,B\,X$ are A" (which generalizes the case "$Q\,X$ are A" by considering that the set to which the quantifier applies is itself fuzzy) where X is a (crisp) referential and A and B are fuzzy predicates. Examples of such fuzzy statements are: "most of the close contributors of Mr. Brown are young", "more than two thirds of the papers authored by Mr. Smith have been published in journals with a high impact factor", etc.

3.2.1. Zadeh's Interpretation

Let X be the usual (crisp) set $\{x_1, x_2, \ldots, x_n\}$ and n the cardinality of X. [14] defines the cardinality of the set of elements of X which satisfy A, as: $\Sigma count(A) = \sum_{i=1}^{n} \mu_A(x_i)$.

The truth degree of the statement "$Q\,B\,X$ are A" (with Q relative) is

then given by:

$$\mu(\mathcal{Q}\,B\,X\,are\,A) = \mu_{\mathcal{Q}}\left(\frac{\Sigma count(A \cap B)}{\Sigma count(B)}\right)$$
$$= \mu_{\mathcal{Q}}\left(\frac{\sum_{x \in X} \top(\mu_A(x),\,\mu_B(x))}{\sum_{x \in X} \mu_B(x)}\right) \tag{4}$$

where $\top$ denotes a triangular norm (e.g., the minimum).

One may notice, however, that a large number of elements with a small degree $\mu_A(x)$ has a same effect as a small number of elements with a high degree $\mu_A(x)$, due to the definition of $\Sigma count$. This is a drawback inasmuch as one cannot distinguish between situations that are quite different.

3.2.2. Interpretation Based on the OWA Operator

In [15], Yager suggests to compute the truth degree of statements of the form "$\mathcal{Q}\,B\,X\,are\,A$" by an OWA aggregation of the implication values $\mu_B(x) \rightarrow_{KD} \mu_A(x)$ where $\rightarrow_{KD}$ denotes Kleene-Dienes implication ($a \rightarrow_{KD} b = \max(1 - a,\,b)$).

Let $X = \{x_1,\,\ldots,\,x_n\}$ such that $\mu_B(x_1) \leq \mu_B(x_2) \leq \ldots \leq \mu_B(x_n)$ and $\sum_{i=1}^n \mu_B(x_i) = d$. The weights of the OWA operator are defined by: $w_i = \mu_{\mathcal{Q}}(S_i) - \mu_{\mathcal{Q}}(S_{i-1})$, with $S_i = \sum_{j=1}^i \frac{\mu_B(x_j)}{d}$ and $S_0 = 0$. The implication values are denoted by $c_i = \max(1 - \mu_{B(x)}, \mu_{A(x)})$ and ordered decreasingly such that $c_1 \geq c_2 \geq \ldots \geq c_n$.

Finally:

$$\mu(\mathcal{Q}\,B\,X\,are\,A) = \sum_{i=1}^n w_i \times c_i. \tag{5}$$

It has been shown in [5] that the OWA-based interpretation of a fuzzy quantified statement corresponds to using a Choquet integral. This interpretation, inasmuch as it does not use the Sigma-count, obviates the drawback mentioned at the end of the previous subsection.

4. Related Work

Fuzzy quantified queries have been studied by several authors in a relational database context. Kacprzyk and Zadrozny [16] propose to use fuzzy quantifiers as connectives for combining atomic conditions, which makes it

possible for instance to retrieve those items that satisfy *most of* the conditions among $\{c_1, \ldots, c_n\}$. Bosc *et al.* [5] suggest to use fuzzy quantifiers in a *having* clause in order to express a condition on the cardinality of a fuzzy subset of a group, as in "find the departments where *most* of the employees are *well-paid*". Let us also mention that fuzzy quantified statements have also been used to model data summaries, see for instance [17, 18].

In a graph database context, beside data *values*, a new dimension can be exploited that concerns the *structure* of the graph. In [11], 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 (cf. Subsection 3.2). 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 [6, 19]. The authors take as an example a graph database representing hotels and their customers and consider the following fuzzy quantified query:

```
1  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:

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

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

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

which returns a fuzzy set of hotels.

In [19], Castelltort and Laurent 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 [17] did for relational databases. 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

$$truth_1(S) = \mu_\mathcal{Q}\left(count(distinct\ S)/count(distinct\ a)\right) \ \text{and}$$
$$truth_2(S) = \mu_\mathcal{Q}\left(count(distinct\ S)/count(distinct\ a\text{–}[r]\text{–}>(?))\right)$$

where ? denotes an edge type variable and *distinct(S)* gives the number of times a relation of type r appears in the graph between a node of type a and a node of type b. 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). The corresponding Cypher query is:

```
1  MATCH (s:student)-[rents]->(a:apartment)
2  WITH toFloat(count(*)) AS countS
3  MATCH (s1:student)-[rents]->(m)
4  WITH toFloat(count(*)) AS count2
5  RETURN MuMost(countS/count2)
```

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 Statements in FUDGE

In this section, we show how fuzzy quantified statements may be expressed in the FUDGE query language. We first propose a syntactic format for these queries, then we show how they can be efficiently evaluated in practice.

5.1. Syntax of a Fuzzy Quantified Query

In the following, we consider *fuzzy quantified queries* involving fuzzy predicates (beside the quantifier) over *fuzzy* graph databases. The fuzzy quanti-

fied statements considered are of the form"$\mathcal{Q}$ B X are A", where the quantifier $\mathcal{Q}$ is represented by a fuzzy set and denotes either an increasing/decreasing relative quantifier (e.g., *most*) or an increasing/decreasing absolute one (e.g., *at least three*), where B is the fuzzy condition "to be connected (according to a given pattern) to a node x", X is the set of nodes in the graph and A is the fuzzy (possibly compound) condition.

An example of such a statement is: "*most* of the *recent* papers of which x is a *main* author, have been published in a *renowned* database journal".

The general syntactic form of a *fuzzy quantified query* of the form"$\mathcal{Q}$ B X are A" in the FUDGE language is given in Listing 3.

```
1  DEFINE... IN
2  MATCH B(res, x)
3  WITH res HAVING Q(x) ARE A(x)
4  RETURN res
```
Listing 3: Syntax of a fuzzy quantified query

This query contains a list of `DEFINE` clauses for the fuzzy quantifiers and the fuzzy terms declarations, a `MATCH` clause for fuzzy graph pattern selection, a `HAVING` clause for the fuzzy quantified statement definition and a `RETURN` clause for specifying which elements should be returned in the result set. `B(res, x)` denotes the fuzzy graph pattern involving the nodes `res` and `x` and expressing the (possibly fuzzy) conditions in B. `B(res, x)` takes the form of a fuzzy graph pattern expressed *à la Cypher* by P_B `WHERE` C_B, where P_B is the "structural" part of B and C_B its "value-based" part (made of conditions on node and edge variables, see Section 2.4). `A(x)` denotes the fuzzy graph pattern involving the node `x` and expressing the (possibly fuzzy) conditions in A. `A(x)` takes the form of a fuzzy graph pattern expressed *à la Cypher* by P_A `WHERE` C_A.

Example 3. *The query, denoted by $Q_{mostAuthors}$, that consists in "retrieving every author (a) such that most of the recent papers (p) of which he/she is a main author, have been published in a renowned database journal (j)" may be expressed in* FUDGE *as follows (it is assumed that renowned means having a high impact factor):*

```
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 (a:author)-[author_of|ST IS strong]->(p:paper)
4  WHERE p.year IS recent
5  WITH a
```
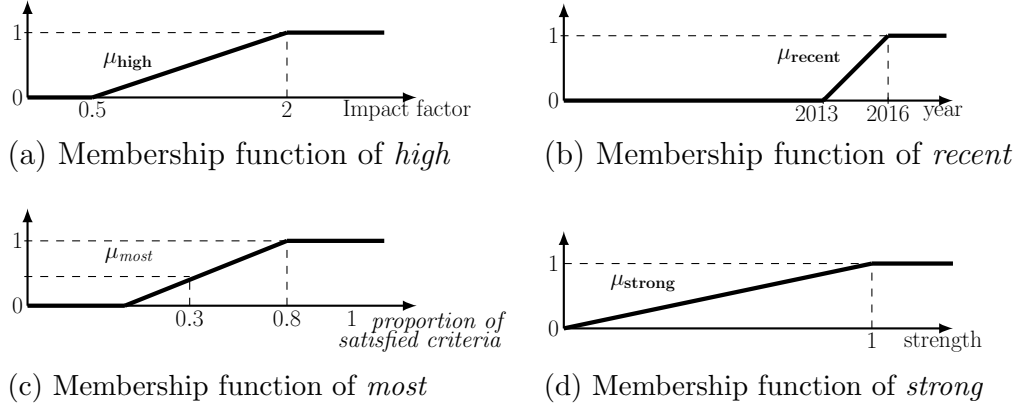
(a) Membership function of *high*

(b) Membership function of *recent*

(c) Membership function of *most*

(d) Membership function of *strong*

Figure 3: Membership functions

```
6   HAVING most(p) ARE ( (p)-[:published]->(j:journal),
335 7     (j)-[:impact_factor]->(i:impact_factor), (j)-[:domain]->(d:dom)
8       WHERE i.value IS high AND d.name="database" )
9   RETURN a
```

Listing 4: Syntax of the fuzzy quantified query $Q_{mostAuthors}$

where the DEFINEQRELATIVEASC *clause defines the fuzzy relative increasing quantifier* most *of Figure 3.(c), and the next* DEFINEASC *clauses define the ascending fuzzy terms* recent, strong *and* high *of Figures 3.(b), 3.(d) and 3(a) respectively. In this query, a corresponds to res, p corresponds to x, lines 3 and 4 correspond to B and lines 6 to 8 correspond to A.*

According to the general syntax introduced in Listing 3, the variable a *instantiates* res *and the variable* p *instantiates* x. ⋄

5.2. Evaluation of a Fuzzy Quantified Query

From a conceptual point of view, the interpretation of such a query can be based on one of the formulas (4) and (5). Its evaluation involves three stages :

1. the compiling of the fuzzy quantified query Q into a crisp query denoted by $Q_{derivedBoolean}$,

2. the interpretation of the crisp query $Q_{derivedBoolean}$,

3. the calculation of the satisfaction degrees associated with the answers to $Q_{derivedBoolean}$.

13

Compiling. The compiling stage translates the fuzzy quantified query Q into
a crisp query denoted by $Q_{derivedBoolean}$. This compilation involves two translation steps.

First, Q is transcripted into a derived query $Q_{derived}$ that allows to interpret the fuzzy quantifier embedded in Q. The query $Q_{derived}$, whose general form[1] is given in Listing 5, aims to retrieve the elements of the B part of the initial query, matching the variables `res` and `x`, for which we will then need to calculate the final satisfaction degree. It is obtained by removing the `WITH` and `HAVING` clauses from the initial query and adding the `OPTIONAL MATCH` clause before the fuzzy graph pattern in condition A.

```
1  MATCH B(res, x)
2  OPTIONAL MATCH A(x)
3  RETURN res x I_A I_B
```

Listing 5: Derived query $Q_{derived}$

Such a query retrieves the pairs $\{res, x\}$ that belong to the graph and all the information needed for the calculation of μ_B and μ_A, i.e., the combination of fuzzy degrees associated with relationships and node attribute values involved in `B(res,x)` and in `A(x)`, respectively denoted by I_B and I_A. Listing 6 of Example 4 below presents the derived query associated with the query $Q_{mostAuthors}$.

The evaluation of $Q_{derived}$ is based on the *derivation principle* introduced by [4] in the context of relational databases: $Q_{derived}$ is derived into another query denoted by $Q_{derivedBoolean}$. The derivation translates the fuzzy query into a crisp one by transforming its fuzzy conditions into Boolean ones that select the support of the fuzzy statements. For instance, the fuzzy condition `p.year IS recent` where the fuzzy term *recent* is defined as `DEFINEAsc recent AS (2013,2016)` is derived into the Boolean condition `p.year > 2013`. Basically, the derivation principle relies on the following property:

x *somewhat satisfies* $c_1 \wedge \ldots \wedge c_n \Leftrightarrow$
$$(x.A_1 \in support(c_1) \wedge \ldots \wedge x.A_n \in support(c_n))$$

where every c_i denotes a fuzzy atomic condition and A_i denotes the attribute concerned by c_i. Listing 7 of Example 4 below is an illustration of the derivation of the query $Q_{derived}$. The way the derivation principle can be applied to FUDGE queries is detailed in [9].

---

[1]Hereafter, the `DEFINE` clauses are omitted for the sake of simplicity.

Crisp interpretation. The previous compiling stage translates the fuzzy quantified query Q embedding fuzzy quantifiers and fuzzy conditions into a crisp query $Q_{derivedBoolean}$, whose interpretation is the classical crisp one.

For the sake of simplicity, we consider in the following that the result of $Q_{derived}$, denoted by $[\![Q_{derived}]\!]$, is made of the quadruples $(\texttt{res}_\texttt{i}, \texttt{x}_\texttt{i}, \mu_{Bi}, \mu_{Ai})$
380 matching the query.

Final result calculation. The last stage of the evaluation calculates the satisfaction degrees μ_B and μ_A according to I_B and I_A. If the optional part does not match a given answer, then $\mu_A = 0$. The answers of the initial fuzzy quantified query Q (involving the fuzzy quantifier $\mathcal{Q}$) are answers of
385 the query $Q_{derived}$ derived from Q and the final satisfaction degree associated with each element e can be calculated according to the two different interpretations mentioned earlier in section 3. Hereafter, we illustrate this using [14] and [15]'s approaches (which are the most commonly used).

Following Zadeh's Sigma-count-based approach (cf. Subsection 3.2.1) we
390 have:

$$\mu(e) = \mu_{\mathcal{Q}} \left(\frac{\sum_{\{(res_i, x_i, \mu_{B_i}, \mu_{A_i}) \in [\![Q_{derived}]\!] | res_i = e\}} \min(\mu_{A_i}, \ \mu_{B_i})}{\sum_{\{(res_i, x_i, \mu_{B_i}, \mu_{A_i}) \in [\![Q_{derived}]\!] | res_i = e\}} \mu_{B_i}} \right) \quad (6)$$

In this formula, μ_{A_i} and μ_{B_i} denote the satisfaction degrees of element $res_i = e$ with respect to conditions A and B respectively.

In the case of a fuzzy absolute quantified query, the final satisfaction degree associated with each element e is simply

$$\mu(e) = \mu_{\mathcal{Q}} \left(\sum_{\{(res_i, x_i, \mu_{B_i}, \mu_{A_i}) \in [\![Q_{derived}]\!] | res_i = e\}} \mu_{A_i} \right) .$$

Example 4. *Let us consider the fuzzy quantified query $Q_{mostAuthors}$ of Listing 4. We evaluate this query according to the fuzzy data graph $\mathcal{DB}$ of Fig-*
395 *ure 1. In order to interpret $Q_{mostAuthors}$, we first derive the following query $Q_{derived}$ from $Q_{mostAuthors}$, that retrieves "the authors (`a`) who are main authors of at least one recent paper (`p`) (corresponds to `B(a,p)` in lines 1 and 2) possibly (`OPTIONAL`) published in a renowned database journal (corresponds to `A(p)` in lines 3 to 5)".*

```
400  1  MATCH (a:author)-[author_of|ST IS strong]->(p:paper)
     2  WHERE p.year IS recent
     3  OPTIONAL MATCH (p)-[:published]->(j:journal),
     4    (j)-[:impact_factor]->(i:impact_factor), (j)-[:domain]->(d:dom)
     5    WHERE i.value IS high AND d.name="database"
405  6  RETURN a p μ_A μ_B
```

<div align="center">Listing 6: Query $Q_{derived}$ derived from $Q_{mostAuthors}$</div>

Then, we evaluate the Cypher query $Q_{derivedBoolean}$ given in Listing 7, derived from the FUDGE non-quantified query $Q_{derived}$ of Listing 6.

```
     1  MATCH fudge_p0 = (a:author)-[:author_of]->(p:paper)
     2  WITH REDUCE(min=1.0, edge IN relationships(fudge_p0)|
410  3  case when edge.fdegree<min
     4  then edge.fdegree else min end) AS fudge_p0,a AS a,p AS p
     5  WHERE fudge_p0>0.0 AND p.year>2013
     6  OPTIONAL MATCH (p)-[:published]->(j:journal),
     7    (j)-[:impact_factor]->(i:impact_factor), (j)-[:domain]->(d:domain)
415  8    WHERE i.value>0.5 AND d.name='database'
     9  RETURN a p μ_A μ_B
```

<div align="center">Listing 7: Query $Q_{derivedBoolean}$ derived from $Q_{derived}$</div>

This query returns a list of author (a) with their papers (p), satisfying the conditions of query $Q_{derived}$, along with their respective satisfaction degrees. Here, μ_B is $\min(\mu_{strong}(\rho_{author}(a, p)), \mu_{recent}(p.year))$ and μ_A is $\mu_{high}(i.value)$.

Let us consider $[\![Q_{derived}(a)]\!]$ the set of answers of the query $Q_{derived}$ for a given author a. The set $[\![Q_{derived}(a)]\!]$ provides a list of papers with their respective satisfaction degrees. This result set is of the form $[\![Q_{derived}(a)]\!] = \{((\mu_B, \mu_A)/p_1), ..., ((\mu_B, \mu_A)/p_n)\}$.

For the running example, $Q_{derived}$ returns the four answers {Peter, Maria, Claudio, Michel}. The authors Andreas, Susan and Bazil do not belong to the result of $Q_{mostAuthors}$ because Susan has not written a journal paper yet and Andreas and Bazil do not have a recent paper.

For the running example, we then have

$[\![Q_{derived}(Peter)]\!] = \{((0.2, 1)/IJAR14\_p)\}$,
$[\![Q_{derived}(Maria)]\!] = \{((0.33, 1)/IJAR14\_p), ((0.6, 0.33)/IJIS16\_p)\}$,
$[\![Q_{derived}(Claudio)]\!] = \{((0.33, 1)/IJAR14\_p), ((0.3, 0.07)/IJUFK15\_p)\}$ and
$[\![Q_{derived}(Michel)]\!] = \{((0.3, 0.07)/IJUFK15\_p)\}$.

Finally, assuming for the sake of simplicity that $\mu_{most}(x) = x$, the final result of the query $Q_{mostAuthors}$ evaluated on $\mathcal{DB}$ using Formula 6 is

$\{\mu(Peter) = \mu_{most}(\frac{0.2}{0.2}) = 1,\ \mu(Maria) = \mu_{most}(\frac{0.66}{0.93}) = 0.71,\ \mu(Claudio) = \mu_{most}(\frac{0.4}{0.63}) = 0.63,\ \mu(Michel) = \mu_{most}(\frac{0.07}{0.3}) = 0.23\}.\diamond$

Using Yager's OWA-based approach (cf. subsection 3.2.2), for each element e returned by $Q_{derived}$ we calculate

$$\mu(e) = \sum_{\{(res_i, x_i, \mu_{Bi}, \mu_{Ai}) \in [\![Q_{derived}]\!] | res_i = e\}} w_i \times c_i. \tag{7}$$

Example 5. *In order to calculate $\mu(\text{Maria})$ from $Q_{derived}$, let us consider B (resp. A) the set of satisfaction degrees corresponding to condition B (resp. A) of element* Maria *as follows B={0.33/IJAR14, 0.6/IJIS16} and A={1/IJAR14, 0.33/IJIS16}. We have $d = 0.93$ and:*

$$S_{IJAR14} = \frac{0.33}{0.93} = 0.35,\ \ and\ \ S_{IJIS16} = \frac{0.33 + 0.6}{0.93} = 1.$$

Then, with $\mu_{most}(x) = x$, we get $\mu_{\mathcal{Q}}(S_{IJAR14}) = 0.35$ and $\mu_{\mathcal{Q}}(S_{IJIS16}) = 1$. Therefore, the weights of the OWA operator are:

$W_1 = \mu_{\mathcal{Q}}(S_{IJAR14}) - \mu_{\mathcal{Q}}(S_0) = 0.35$ *and* $W_2 = \mu_{\mathcal{Q}}(S_{IJIS16}) - \mu_{\mathcal{Q}}(S_{IJAR14}) = 0.65$.

The implication values are:

$c_{IJAR14} = \max(1 - 0.33, 1) = 1,\ \ and\ \ c_{IJIS16} = \max(1 - 0.6, 0.33) = 0.4.$

Thus, $c_1 = 1$ and $c_2 = 0.4$. Finally, we get:

$$\mu(\text{Maria}) = 0.35 \times 1 + 0.65 \times 0.4 = 0.35 + 0.26 = 0.61.$$

440 *Lastly, the final result of the query $Q_{mostAuthors}$ evaluated on $\mathcal{DB}$, given by Formula 7, is: $\{\mu(Peter) = 1,\ \mu(Claudio) = 0.84,\ \mu(Michel) = 0.7,\ \mu(Maria) = 0.61\}.\ \diamond$*

6. About Query Processing

For the implementation of these quantified queries, we updated the SUGAR
445 software of [8, 20], which is a software add-on layer that implements the
FUDGE language over the *Neo4j* graph DBMS. This software efficiently evaluates FUDGE queries that contain fuzzy preferences, but its initial version did not support fuzzy quantified statements.

17

The current work is an extended version in which we also consider Yager's OWA-based interpretation (whereas the initial version only used Zadeh's interpretation) and a single fuzzy query is derived for the evaluation of the quantified query instead of three queries as initially proposed in [7] where a fuzzy quantified query Q was indeed derived into:

- a first query Q_1 aimed to retrieve the elements matching the variable x, for which one needs to calculate a satisfaction degree. Query Q_1 was obtained by removing the `WITH` and `HAVING` clauses from the initial query;

- a second (resp. third) derived query, denoted by $Q_2(e)$ (resp. $Q_3(e)$) aimed to compute the degree corresponding to the denominator (resp. numerator) of Equation 6 for every element e returned by Q_1.

The sugar software basically consists of two modules, which implement the *Compiling* and *Final result calculation* stages defined in Section 5.2. These modules interact with a Neo4j crisp engine, which implements the *Crisp implementation* stage defined in Section 5.2. Figure 4 illustrates this architecture.

In a pre-processing step, the *Query compiler* module produces (i) the query-dependent functions that allow us to compute μ_B, μ_A and μ, for each returned answer, according to the chosen interpretation, and (ii) the (crisp) Cypher query $Q_{derivedBoolean}$, which is then sent to the crisp Neo4j engine for retrieving the information needed to calculate μ_B and μ_A.

In a post-processing step, the *Score calculator* module performs a grouping (according to the `WITH` clause of the initial query) of the elements, then calculates μ_B, μ_A and μ for each returned answer and finally ranks the answers.

Remark 1. *Since repeating the definition of the fuzzy terms in every query may be tedious for the user, an improvement would be to store the definitions of the fuzzy terms in user profiles that could be accessed by the query compiler.*

For quantified queries of the type introduced in the previous sections (i.e. using relative quantifiers), the principle is to first evaluate the fuzzy query $Q_{derivedBoolean}$ derived from the original query. For each element $x \in [\![Q_{derivedBoolean}]\!]$, we return the satisfaction degrees related to conditions A and B, denoted respectively by μ_A and μ_B. The final satisfaction degree μ
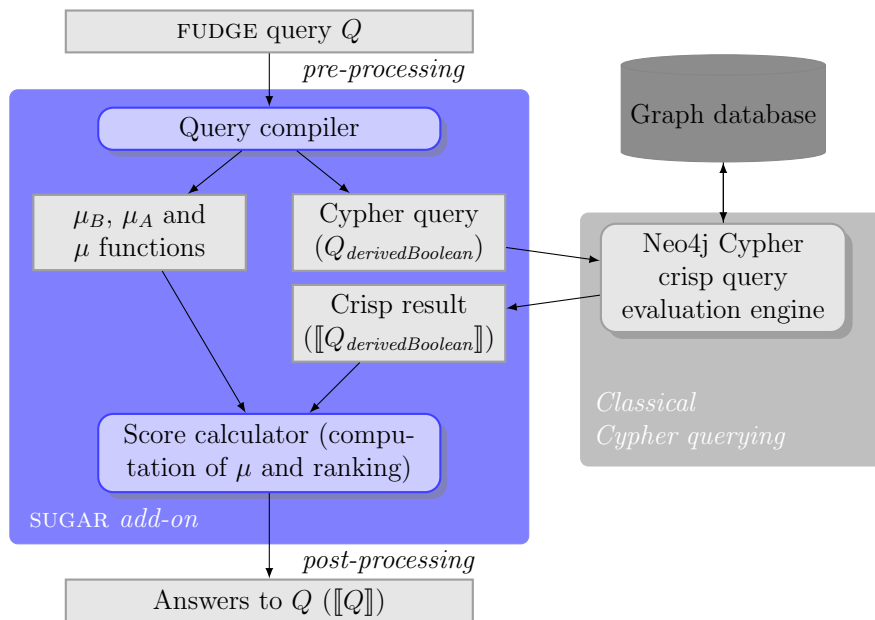
18

Figure 4: SUGAR software architecture

can be calculated according to Formulas (4) or (5) using the values of μ_B and μ_A. [14]'s approach and [15]'s OWA-based approach have been implemented
485 and the choice of the interpretation to be used is made through the system configuration tool. Finally, a set of answers ranked in decreasing order of their satisfaction degree is returned.

As a proof-of-concept of the proposed approach, the FUDGE prototype is available at www-shaman.irisa.fr/fudge-prototype. A screenshot of this proto-
490 type is shown in Figure 5 which contains the final result of the evaluation of the query $Q_{mostAuthors}$ of Example 3. The GUI is composed of two frames i) a central frame for visualizing the graph and the results of a query and ii) an input field frame (placed under the central one), for entering and running a FUDGE query.

495 **7. Experimental Results**

In order to confirm the effectiveness and efficiency of the approach, we carried out some experiments on a computer running on Windows 7 (64 bits) with 8 Gb of RAM. We considered four queries (based on the topology of [21]) with various forms of condition A.
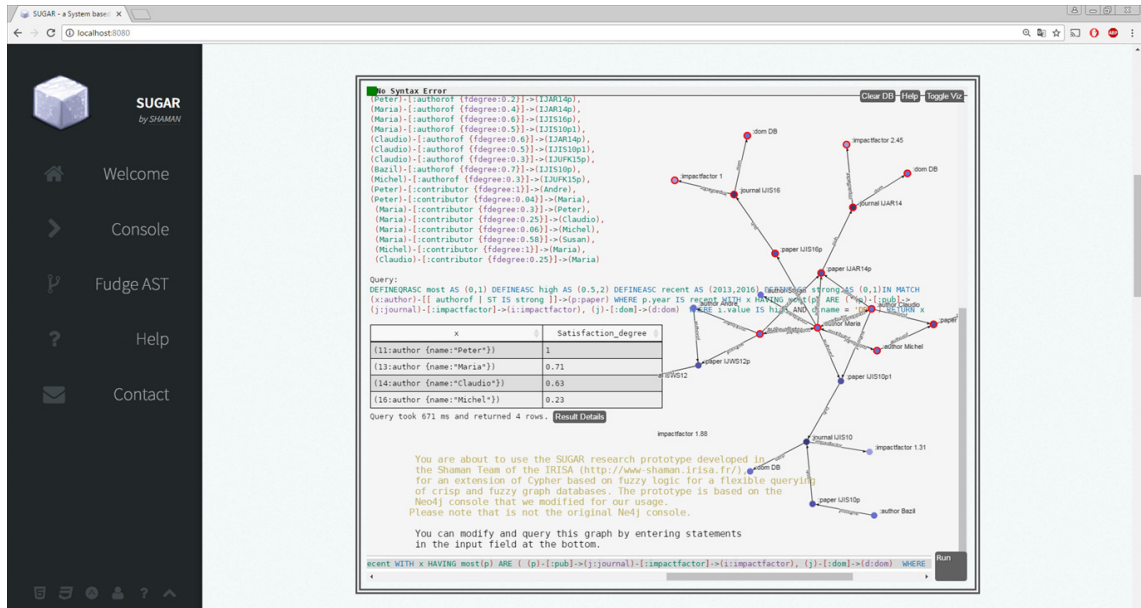
19

Figure 5: Screenshot of the FUDGE prototype

The first query Q_1, where A is an *adjacency pattern*, aims to find the authors such that *most* of the *recent* papers of which they are *main* authors, have been published in a journal. In the following, we omit the `DEFINE` clause of the queries in order to focus on the most interesting part.

```
1  MATCH (a:author)-[author_of|ST IS strong]->(p:paper)
2  WHERE p.year IS recent
3  WITH a
4  HAVING most(p) ARE ( (p)-[:published]->(j:journal) )
5  RETURN a
```

Listing 8: Fuzzy quantified query with adjacency pattern Q_1

The second query Q_2, where A is a *reachability pattern* involving *fixed length path*, aims to find the authors such that *most* of the *recent* papers of which they are *main* authors, have been published in a ranked journal.

```
1  MATCH (a:author)-[author_of|ST IS strong]->(p:paper)
2  WHERE p.year IS recent
3  WITH a
4  HAVING most(p) ARE ( (p)-[:published]->(j:journal),
5                       (j)-[:impact_factor]->(i:impact_factor) )
```

20

Table 1: Fuzzy graph datasets

Dataset	Size
DB_1	700 nodes & 1447 edges
DB_2	2100 nodes & 4545 edges
DB_3	3500 nodes & 7571 edges
DB_4	4900 nodes & 10494 edges

```
6   RETURN a
```
Listing 9: Fuzzy quantified query with reachability pattern Q_2

The third query Q_3, where A is a *reachability pattern* involving *regular simple path*, aims to find the authors a such that *most* of the *recent* papers of which they are *main* authors, have been co-authored by another author b who is a contributor (not necessarily direct) of *Claudio*.

```
1   MATCH (a:author)-[author_of|ST IS strong]->(p:paper)
2   WHERE p.year IS recent
3   WITH a
4   HAVING most(p) ARE ( (b:author)-[:author_of]->(p),
5                         (b)-[:contributor*]->(c:author)
6                   WHERE c.name="Claudio" )
7   RETURN a
```
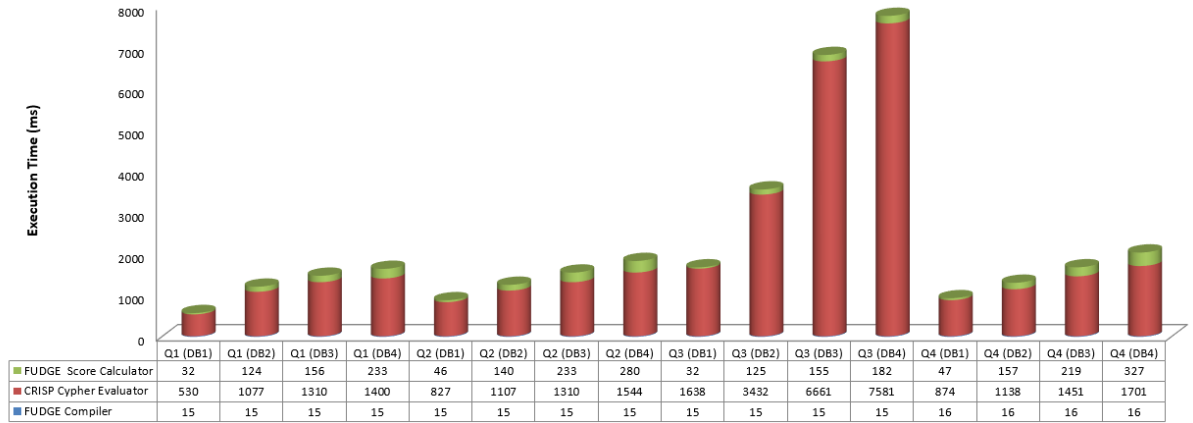Listing 10: Fuzzy quantified with reachability query Q_3

The fourth query Q_4 is the query $Q_{mostAuthors}$ of Listing 4, where A is a *pattern matching*.

Our experiments have been performed on a graph database inspired from DBLP containing crisp (e.g., published) and fuzzy edges (e.g., contributor). Four database sizes have been considered, see Table 1.

The results of the processing of these queries over the datasets from Table 1 are depicted in Figure 6 where Figure 6.(a) (resp. Figure 6.(b)) presents the execution time in milliseconds using Zadeh's interpretation (resp. Yager's OWA-based interpretation).

A crucial result is that the processing time taken by the *compiling* step and the *score calculation* step, which are directly related to the introduction of flexibility into the query language, are very strongly dominated by the time taken by the *crisp Cypher evaluator*.

Moreover, the FUDGE *compiling* step takes so little time compared to the other two steps that it cannot even be seen in Figure 6. This time remains

(a) Zadeh's interpretation over different size of DB

	Q1 (DB1)	Q1 (DB2)	Q1 (DB3)	Q1 (DB4)	Q2 (DB1)	Q2 (DB2)	Q2 (DB3)	Q2 (DB4)	Q3 (DB1)	Q3 (DB2)	Q3 (DB3)	Q3 (DB4)	Q4 (DB1)	Q4 (DB2)	Q4 (DB3)	Q4 (DB4)
FUDGE Score Calculator	32	124	156	233	46	140	233	280	32	125	155	182	47	157	219	327
CRISP Cypher Evaluator	530	1077	1310	1400	827	1107	1310	1544	1638	3432	6661	7581	874	1138	1451	1701
FUDGE Compiler	15	15	15	15	15	15	15	15	15	15	15	15	16	16	16	16

(b) Yager's interpretation over different size of DB

	Q1 (DB1)	Q1 (DB2)	Q1 (DB3)	Q1 (DB4)	Q2 (DB1)	Q2 (DB2)	Q2 (DB3)	Q2 (DB4)	Q3 (DB1)	Q3 (DB2)	Q3 (DB3)	Q3 (DB4)	Q4 (DB1)	Q4 (DB2)	Q4 (DB3)	Q4 (DB4)
FUDGE Score Calculator	47	141	203	296	48	157	267	312	47	156	203	240	75	187	312	406
CRISP Cypher Evaluator	562	1030	1295	1404	827	1076	1343	1560	1654	3322	6599	7704	901	1139	1482	1700
FUDGE Compiler	15	15	15	15	15	15	15	15	15	15	15	15	16	16	16	16

Figure 6: Experimental results

almost constant and is independent on the size of the dataset while slightly increasing in the presence of complex patterns or fuzzy conditions. As to the *score calculation* step, it represents around 9% of the time needed for evaluating a crisp Cypher query. The time used for calculating the final satisfaction degree is of course dependent on the size of the result set and the nature of the patterns.

Finally, these results show that introducing *fuzzy quantified statements* into a FUDGE query entails a reasonable increase of the overall processing time in the case of selection graph pattern queries. It represents, in the worst case, around 11% of the time needed for evaluating a fuzzy quantified FUDGE query. Of course, as can be seen in Figures 6.(a) and Figure 6.(b), some queries are more costly than others, depending on the complexity of the graph pattern involved, as in the Boolean case.

8. 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. We performed some experiments in order to study its performances. The results of these experiments show that the cost of dealing with fuzzy quantification in a query is reasonable w.r.t. the cost of the overall evaluation.

As a future work, we first 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 papers 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.

References

[1] R. Angles, C. Gutierrez, Survey of graph database models, ACM Comput. Surv. 40 (1) (2008) 1–39.

[2] K. Stefanidis, G. Koutrika, E. Pitoura, A survey on representation, composition and application of preferences in database systems, ACM Trans. Database Syst. 36 (3) (2011) 19.

[3] V. Tahani, A conceptual framework for fuzzy query processing - A step toward very intelligent database systems, Inf. Process. Manage. 13 (5) (1977) 289–303.

[4] O. Pivert, P. Bosc, Fuzzy Preference Queries to Relational Databases, Imperial College Press, London, UK, 2012.

[5] P. Bosc, L. Liétard, O. Pivert, Quantified statements and database fuzzy querying, in: P. Bosc, J. Kacprzyk (Eds.), Fuzziness in Database Management Systems, Physica Verlag, 1995, pp. 275–308.

[6] A. Castelltort, A. Laurent, Fuzzy queries over NoSQL graph databases: Perspectives for extending the cypher language, in: Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU), 2014, pp. 384–395.

[7] O. Pivert, O. Slama, V. Thion, Fuzzy quantified structural queries to fuzzy graph databases, in: International Conference on Scalable Uncertainty Management (SUM), Springer, 2016, pp. 260–273.

[8] O. Pivert, V. Thion, H. Jaudoin, G. Smits, On a fuzzy algebra for querying graph databases, in: Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI), Limassol, Cyprus, 2014, pp. 748–755.

[9] O. Pivert, G. Smits, V. Thion, Expression and efficient processing of fuzzy queries in a graph database context, in: Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2015.

[10] A. Rosenfeld, Fuzzy graphs, in: Fuzzy Sets and their Applications to Cognitive and Decision Processes, Academic Press, 1975, pp. 77–97.

[11] R. R. Yager, Social network database querying based on computing with words, in: O. Pivert, S. Zadrożny (Eds.), Flexible Approaches in Data, Information and Knowledge Management, Vol. 497 of Studies in Computational Intelligence, Springer, 2013, pp. 241–257.

[12] The Neo4j Developer Manual v3.2 (2017).

[13] Neo4j web site, `www.neo4j.org` (consulted in 2017).

[14] L. A. Zadeh, A computational approach to fuzzy quantifiers in natural languages, Computing and Mathematics with Applications 9 (1983) 149–183.

[15] R. R. Yager, On ordered weighted averaging aggregation operators in multicriteria decisionmaking, IEEE Transactions on Systems, Man, and Cybernetics 18 (1) (1988) 183–190.

[16] J. Kacprzyk, S. Zadrożny, A. Ziólkowski, FQUERY III +:a "human-consistent" database querying system based on fuzzy logic with linguistic quantifiers, Inf. Syst. 14 (6) (1989) 443–453.

[17] D. Rasmussen, R. R. Yager, Summary SQL - A fuzzy tool for data mining, Intell. Data Anal. 1 (1-4) (1997) 49–58.

[18] J. Kacprzyk, S. Zadrożny, Linguistic database summaries and their protoforms: towards natural language based knowledge discovery tools, Information Sciences 173 (4) (2005) 281–304.

[19] A. Castelltort, A. Laurent, Extracting fuzzy summaries from NoSQL graph databases, in: Proceedings of the 11th International Conference on Flexible Query Answering Systems (FQAS), Vol. 400 of Advances in Intelligent Systems and Computing, Springer, Cracow, Poland, 2015, pp. 189–200.

[20] O. Pivert, O. Slama, G. Smits, V. Thion, SUGAR: A Graph Database Fuzzy Querying System, in: Proceedings of the IEEE International Conference on Research Challenges in Information Science (RCIS) Demos, Grenoble, France, 2016.

[21] R. Angles, A comparison of current graph database models, in: Proceedings of the 28th IEEE International Conference on Data Engineering Workshops (ICDEW), IEEE, 2012, pp. 171–177.