On a Fuzzy Algebra for Querying Graph Databases

Olivier Pivert, Virginie Thion, Hélène Jaudoin, Grégory Smits

On a Fuzzy Algebra for Querying Graph Databases

Olivier Pivert, Virginie Thion, Hélène Jaudoin, Grégory Smits
IRISA – University of Rennes 1 – Lannion, France
pivert@enssat.fr, virginie.thion@irisa.fr, jaudoin@enssat.fr, gregory.smits@irisa.fr

***Abstract*—This paper proposes a notion of fuzzy graph database and describes a fuzzy query algebra that makes it possible to handle such database, which may be fuzzy or not, in a flexible way. The algebra, based on fuzzy set theory and the concept of a fuzzy graph, is composed of a set of operators that can be used to express preference queries on fuzzy graph databases. The preferences concern i) the content of the vertices of the graph and ii) the structure of the graph. In a similar way as relational algebra constitutes the basis of SQL, the fuzzy algebra proposed here underlies a user-oriented query language and an associated tool implementing this language that are also presented in the paper.**

***Keywords*-Graph Database, Query Algebra, Fuzzy theory**

I. Introduction

Much work has been done about fuzzy querying of *relational* databases, cf. for instance [1] or [2], which led in particular to a fuzzy extension of the SQL language, called SQLf [3]. 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, a new concept has started to attract a lot of attention in the database world, namely that of *graph databases* (see eg. [4], [5], [6], [7], [8], [9] and [10]), whose basic purpose is to efficiently manage networks of entities where each node is described by a set of characteristics (for instance a set of attributes), and each edge represents a link between entities. Such a database model has many potential applications, e.g. for modeling social networks, RDF data, cartographic databases, bibliographic databases, etc.

Graph databases raise new challenges in terms of flexible querying since two aspects may be involved in the preferences that a user may express: i) the content of the nodes and ii) the structure of the graph itself. Furthermore, graph database management systems still lack query languages with a clear syntax and semantics [11]. Our work tackles these two problems as we aim to outline a fuzzy algebra suited to querying graph databases in a flexible way, for graph databases that may be fuzzy or not.

Our contributions are: (i) a formal definition of fuzzy graph databases and a fuzzy algebra for querying fuzzy or crisp graph databases and (ii) a language based on this algebra with an associated prototype software.

The remainder of the paper is structured as follows. Section II presents some background notions about graph databases, fuzzy set theory and fuzzy graphs. Section III describes the main elements that may be involved in a fuzzy query addressed to a graph database. Section IV is the heart of the paper: it proposes a definition of a fuzzy graph database and presents the fuzzy query algebra. Then Section V discusses of a concrete language based on the algebra, and presents an tool implementing this language. Related work is discussed in Section VI. Section VII recalls the contributions and outlines perspectives for future work.

II. Background notions

A. 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 [12]. Among the existing systems, let us mention AllegroGraph [13], InfiniteGraph [14], Neo4j [15] and Sparksee [16]. There are different models for graph databases (see [12] 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*).

B. Fuzzy sets and fuzzy graphs

Fuzzy set theory was introduced by Zadeh [17] for modeling classes or sets whose boundaries are not clear-cut. For such objects, the transition between full membership and full mismatch is gradual rather than crisp. Typical examples of such fuzzy classes are those described using adjectives of the natural language, such as *young, cheap, fast*, etc. Formally, a fuzzy set F on a referential U is characterized by a membership function $\mu_F : U \rightarrow [0, 1]$ where $\mu_F(u)$ denotes the grade of membership of u in F. In particular, $\mu_F(u) = 1$ reflects full membership of u in F, while $\mu_F(u) = 0$ expresses absolute non-membership. When $0 < \mu_F(u) < 1$, one speaks of partial membership. Two crisp sets are of particular interest when defining a fuzzy set F: the core $C(F) = \{u \in U \,|\, \mu_F(u) = 1\}$, which gathers the *prototypes* of F, and the support $S(F) = \{u \in U \mid \mu_F(u) > 0\}$.

In practice, the membership function associated with F is often of a trapezoidal shape. Then, F is expressed by the quadruplet $(A,\ B,\ a,\ b)$ where $C(F) = [A,\ B]$ and $S(F) = [A - a,\ B + b]$, see Fig. 1.

The α-cut of a fuzzy set F, denoted by F^α, is an ordinary set of elements whose satisfaction degree is at least equal to

α: $F^\alpha = \{u \in U \mid \mu_F(u) \geq \alpha\}$. Thus, $C(F)$ and $S(F)$ are two particular α-cuts of F where α is respectively equal to 1 and 0^+.

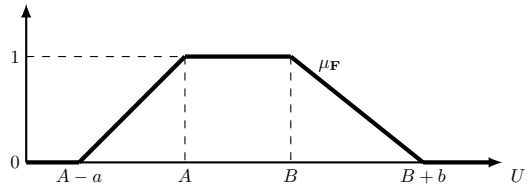

Figure 1. Trapezoidal membership function

Let F and G be two fuzzy sets on the universe U, we say that $F \subseteq G$ iff $\mu_F(u) \leq \mu_G(u)$, $\forall u \in U$. The complement of F, denoted by F^c, is defined by $\mu_{F^c}(u) = 1 - \mu_F(u)$. Furthermore, $F \cap G$ (resp. $F \cup G$) is defined the following way: $\mu_{F \cap G}(u) = \min(\mu_F(u), \mu_G(u))$ (resp. $\mu_{F \cup G}(u) = \max(\mu_F(u), \mu_G(u))$). As usual, the logical counterparts of the theoretical set operators $\cap$, $\cup$ and complementation operator correspond respectively to the conjunction $\wedge$, disjunction $\vee$ and negation $\neg$. See [18] for more details.

A *graph* 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 [19], [20], where the edge $(x, y) \in V \times V$ has weight or strength $\rho(x, y) \in [0, 1]$. This degree may express the "intensity" of any kind of gradual relation between two nodes.

Remark 1: The graph may be fuzzy from the start — relation ρ is given — or be made fuzzy. It may also involve a dynamical aspect. For instance, in a Twitter-like network, $\rho(x, y)$ may be defined as a function of the number of posts by y that x has forwarded.

As noted in [21], the fuzzy relation ρ may be viewed as a fuzzy subset on $V \times V$, which allows us to use much of the formalism of fuzzy sets. For example, we can say that $\rho_1 \subseteq \rho_2$ if $\forall(x, y)$, $\rho_1(x, y) \leq \rho_2(x, y)$. Some notable properties that can be associated with fuzzy relations are reflexivity ($\rho(x, x) = 1, \forall x$), symmetry ($\rho(x, y) = \rho(y, x)$), transitivity ($\rho(x, z) \geq \max_y \min(\rho(x, y), \rho(y, z))$).

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 [21], we define ρ^0 to be s. t. $\rho^0(x, y) = 0$, $\forall(x, y)$.

If ρ is reflexive then $\rho^{k_2} \supseteq \rho^{k_1}$ for $k_2 > k_1$. On the other hand, if ρ is transitive, it can be shown that $\rho^{k_2} \subseteq \rho^{k_1}$ if $k_2 > k_1$. From this, we see that if ρ is reflexive and transitive then $\rho^{k_2} = \rho^{k_1}$ for all k_1 and $k_2 \neq 0$.

Remark 2: Fuzzy graphs as defined above may be generalized to the case where a fuzzy set of vertices is considered. Then, denoting by F the fuzzy subset of V considered, the corresponding fuzzy graph is defined as (V, F, ρ_F). In this case, we let ρ_F be a relation on V defined as $\rho_F(x, y) = \min(\rho(x, y), \mu_F(x), \mu_F(y))$ where μ_F denotes the membership function attached to F. In the following, we only consider the simple case of a crisp set of vertices. If ρ is symmetric, we shall say that (V, ρ) is an undirected graph. Otherwise, we shall refer to (V, ρ) as a directed graph.Without loss of generality, we consider directed graphs in the following.

III. Fuzzy preferences on graph DBs

In this section, we describe the main elements that may appear in a fuzzy query addressed to a graph database. Two types of preferences have to be considered: those on content and those on structure.

1) Preferences on the node contents: The idea is to express flexible conditions about the attributes associated with nodes and/or vertices of the graph. An example is: "find the people who are *young*, *highly educated*, and live in *Eastern Europe*" (assuming that each node contains information about the age, education level, address, etc., of the person it corresponds to). Compound conditions may also be expressed using a large range of fuzzy connectives. We do not get into more detail as this aspect has been thoroughly studied in a relational context [2].

2) Preferences on the graph structure : Hereafter, we describe the concepts of fuzzy graph theory that appear the most useful in a perspective of graph database querying. We denote a *fuzzy graph* by $G = (V, \rho)$.

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

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

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 $\rho^\infty(x, y)$. An algorithm to compute ρ^∞ is given in [22].

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 a concept defined by

Rosenfeld [19] as follows:

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

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

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

It is the length of the shortest path from x to y. It can be shown that δ is a metric [19], i.e., $\delta(x,\, x) = 0$, $\delta(x,\, y) = \delta(y,\, x)$, and $\delta(x,\, z) \leq \delta(x,\, y) + \delta(y,\, z)$.

α-cut of a relation. — It is defined as follows: $\rho^\alpha = \{(x,\, y) \mid \rho(x,\, y) \geq \alpha\}$ where $\alpha \in [0^+, 1]$. Note that ρ^α is a crisp relation.

3) Preference combination: Different types of aggregation may be considered for combining conditions about the content or the structure of the graph: "flat" (min, max, arithmetic mean, etc.), weighted (weighted mean, OWA, quantified proposition, etc, see [23]), or hierarchical.

IV. Fuzzy graph databases and fuzzy algebra

In this section, we define an algebra suited to the flexible handling of fuzzy or crisp graph databases. We first introduce the data model, then the algebra.

A. Data model

In the following, we are interested in fuzzy graph databases where nodes and edges can carry data (e.g. key-value pairs in attributed graphs, see Section II-A). So, we first propose an extension of the definition of a *fuzzy graph* into that of a *fuzzy data graph*.

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 an id, also denoted 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 data value, e.g. a set of key-value pairs, to nodes (resp. edges) of G.

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 $\rho_e : V \times V \rightarrow \{0,\, 1\}$ for all $e \in E$). We then only deal with fuzzy edges and data graph in the following.

In the following, a *graph database* is meant to be a fuzzy data graph. The following example illustrates this notion.

Example 1: Fig. 2 is an example of fuzzy data graph inspired of data stemming from DBLP[1], with some fuzzy edges (with a corresponding degree in brackets), and crisp ones (equivalent to a corresponding degree of `1`). In this example, 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`. In this example, the degree is based on a simple statistical notion, which can be made subtler by fuzzy operations or by the integration of expert knowledge. ⋄

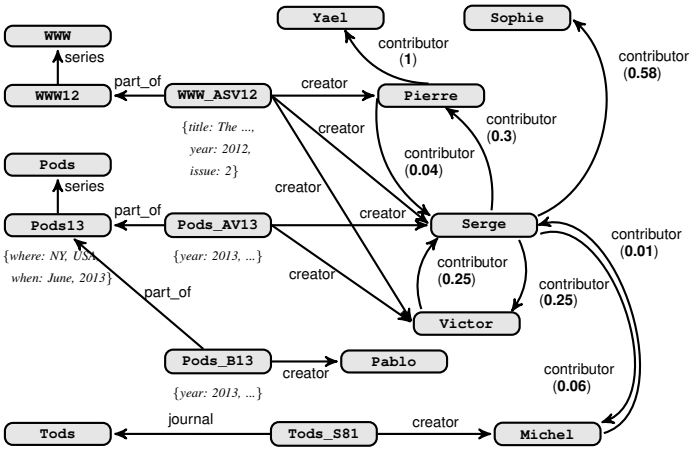

Figure 2. A fuzzy data graph $\mathcal{DB}$ inspired of an excerpt of DBLP data

Nodes are assumed to be typed, as many existing systems offer this facility. If n is a node of V, then $Type(n)$ denotes its type. In $\mathcal{DB}$ (Fig 2), the nodes `WWW12` and `Pods13` are of type *Conference*, the nodes `Pods_AV13`, `Pods_B13`, `Tods_S81`, and `WWW_ASV12` are of type *Article*, the nodes `Pods`, `Tods`, and `WWW` are of type *Serie* and the other nodes are of type *Author*.

B. Algebra

We now move to the definition of a graph algebra allowing the definition of flexible queries, for handling fuzzy or crisp graph databases. Along the lines of relational algebra, the algebra could be the core of a practical language and serve as a basis for describing and optimizing underlying query execution plans. The algebra is partly inspired from graph pattern queries of [24] and from the crisp algebra proposed in [5]. The basic unit of information is the graph.

The first operator is the *selection*, which is based on the concept of *fuzzy graph pattern*, an extension of the crisp graph pattern notion defined in [24] shown to have good properties for a practical implementation. We first introduce the notion of a *fuzzy regular expression* subsequently used for defining a *fuzzy graph pattern*.

Definition 2 (Fuzzy regular expression): A *fuzzy regular expression* is an expression of the form

$$F ::= e \mid F{\cdot}F \mid F \cup F \mid F^* \mid F^{Cond}$$

where

- $e \in E \cup \{\_\}$ denotes an edge labeled by e, with the wildcard symbol denoting any label in E;
- $F{\cdot}F$ denotes a concatenation of expressions;
- $F \cup F$ denotes alternative expressions;

[1] http://www.informatik.uni-trier.de/~ley/db/

- F^* denotes the classical repetition of an expression;
- F^{Cond} denotes paths p satisfying F and the condition $Cond$ where $Cond$ is a boolean combination of atomic formulas of the form: `Prop IS Fterm` where `Prop` is a property defined on p and `Fterm` denotes a predefined or user-defined fuzzy term like `short` (see Fig. 3, which proposes a representation (membership function) associated with the fuzzy term *short*).

In the following, we limit properties to $\{\texttt{ST}, \texttt{Length}\}$ denoting resp. $ST(p)$ (See Equation 1) and $Length(p)$ (See Equation 2). Examples of conditions of this form are `Length IS short` and `ST IS strong`. Notice that Boolean conditions of the form `Property op a` where `Property` is a property on p, `a` is a constant and `op` is a comparison operator ($<, =, ...$) are a special case of fuzzy conditions.

We use the following shortcut notations: giving a fuzzy regular expression f, f^+ is a shortcut for $f{\cdot}f^*$, f^k is a shortcut for $f{\cdot}f{\cdot}\cdots{\cdot}f$ with k occurrences of f and $f^{n,m}$ is a shortcut for $\bigcup_{i=n}^{m} f^i$.

A fuzzy regular expression is said to be *simple* if it is of the form e where $e \in E \cup \{\_\}$ meaning that it explicitly denotes a single edge.

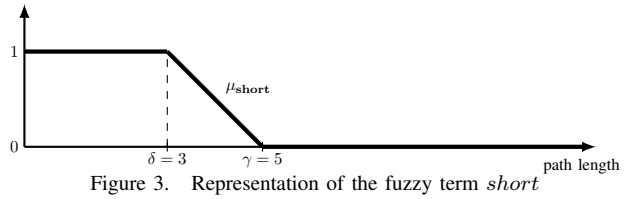

Figure 3. Representation of the fuzzy term *short*

Remark 3: Even if one considers only crisp graphs, the concepts presented above can still be used as arguments of fuzzy conditions (for instance, *short* length) since these concepts are still valid on classical graphs (when $\rho(x) \in \{0, 1\}$).

Definition 3 (Fuzzy regular expression matching): Given a path p and a fuzzy regular expression exp, p *matches* exp *with a satisfaction degree of* $\mu_{exp}(p)$ defined as follows, according to the form of exp (in the following, f, f_1 and f_2 are fuzzy regular expressions):

- If exp is of the form e with $e \in E$ (resp. "_"). If p is of the form $v_1 \xrightarrow{e'} v_1'$ where $e' = e$ (resp. where $e' \in E$) then $\mu_{exp}(p) = 1$ else $\mu_{exp}(p) = 0$.
- If exp is of the form $f_1{\cdot}f_2$. Let P be the set of all pairs of paths (p_1, p_2) s.t. p is of the form p_1p_2. One has $\mu_{exp}(p) = \max_P(min(\mu_{f_1}(p_1), \mu_{f_2}(p_2)))$.
- If exp is of the form $f_1 \cup f_2$. One has $\mu_{exp}(p) = \max(\mu_{f_1}(p), \mu_{f_2}(p))$.
- If exp is of the form f^*. If p is an empty path then $\mu_{exp}(p) = 1$. Otherwise, we denote by P the set of all tuples of paths $(p_1, \cdots, p_n)$ $(n > 0)$ s.t. p is of the form $p_1 \cdots p_n$. One has $\mu_{exp}(p) = \max_P(\min_{i\in[1..n]}(\mu_f(p_i)))$.
- If exp is of the form f^{Cond} where $Cond$ is a (possibly compound) fuzzy condition. One has $\mu_{exp}(p) = \min(\mu_f(p), \mu_{Cond}(p))$ where $\mu_{Cond}(p)$ is the degree of satisfaction of $Cond$ by p.

Not matching is equivalent to matching with a satisfaction degree of 0.

Example 2: The paths represented in Fig. 4 are some paths from the graph database depicted in Fig. 2.

- Expression $e_1 = creator \cdot contributor^+$ is a fuzzy regular expression. All paths p_i ($i \in [1..4]$) of Fig. 4 match e_1 with a satisfaction degree of $\mu_{e_1}(p_i) = 1$.
- Expression $e_2 = (creator \cdot contributor^+)^{\texttt{ST>0.4}}$ is a fuzzy regular expression. Path p_4 is the only one of Fig. 4 that matches e_2 (as strength $ST(p_1) = 0.3$, $ST(p_2) = 0.3$, $ST(p_3) = 0.01$ and $ST(p_4) = 0.58$), with $\mu_{e_2}(p_4) = 1$.
- Expression $e_3 = creator \cdot (contributor^+)^{\texttt{Length IS short}}$, where $short$ is the fuzzy term of Fig. 3, is a fuzzy regular expression. Paths p_1, p_2 and p_4 of Fig. 4 match e_3 with $\mu_{e_3}(p_1) = 0.83$ as $\mu_{short}(1/0.3) = 0.83$ (where $1/0.3$ is the length of path from `Serge` to `Pierre`), $\mu_{e_3}(p_2) = 0.67$ as $\mu_{short}(1/0.3 + 1) = 0.67$ (where $1/0.67$ is the length of the $short$ path from `Serge` to `Yael`) and $\mu_{e_3}(p_4) = 1$ as $\mu_{short}(1/0.58) = 1$. Path p_3 does not match e_3 as $\mu_{short}(1/0.01) = 0$. $\diamond$

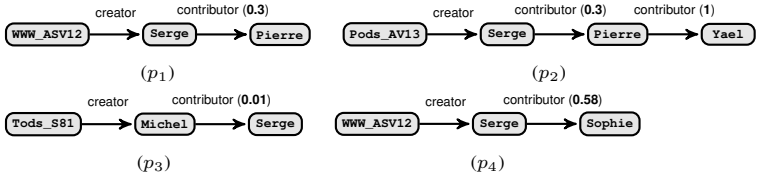

Figure 4. Fuzzy regular expression matching

We then introduce the notion of a *fuzzy graph pattern*, which is a directed crisp graph with conditions on nodes and edges, types on nodes, and where edges are labeled by fuzzy regular expressions that denote paths.

Definition 4 (Fuzzy graph pattern): Let $\mathcal{F}$ be a set of fuzzy terms. A *fuzzy graph pattern* is defined as a sextuple $\mathcal{P} = (V_\mathcal{P}, E_\mathcal{P}, f_e^{path}, f_n^{cond}, f_e^{cond}, f_n^{type})$ where

- $V_\mathcal{P}$ is a finite set of nodes;
- $E_\mathcal{P} \subseteq V_\mathcal{P} \times V_\mathcal{P}$ is a finite set of edges where (u, u') denotes an edge from u to u';
- f_e^{path} is a function defined on $E_\mathcal{P}$ s. t. for each (u, u') in $E_\mathcal{P}$, $f_e^{path}(u, u')$ is a fuzzy regular expression;
- f_n^{cond} is a function defined on $V_\mathcal{P}$ s.t. for each node u, $f_n^{cond}(u)$ is a condition on attributes of u, defined as a combination of atomic formulas of the form `A IS Fterm` where `A` denotes an attribute and `Fterm` denotes a fuzzy term (like eg. `year IS recent`). Again, Boolean predicates of the form `A op a` (where `A` is an attribute, `a` is a constant and `op` is a comparison operator, as in `year > 2012`) are a special case.

- f_e^{cond} is the counterpart of f_n^{cond} for edges. For each (u, u') in $E_{\mathcal{P}}$ for which $f_e^{path}(u, u')$ is simple, f_e^{cond} is the condition on attributes of (u, u'); and
- f_n^{type} is a function defined on $V_{\mathcal{P}}$ s. t. for each node u, $f_n^{type}(u)$ is the type of u.

In the following, we adopt a syntax *à la* CYPHER for graph pattern representation. CYPHER [25] is an intuitive query language inspired from ASCII-art for graph representation, implemented in the Neo4j (crisp) graph database management system [15]. A fuzzy graph pattern expressed *à la* CYPHER consists of a set of expressions of the form `(n1:Type1)-[exp]->(n2:Type2)` or `(n1:Type1)-[e:label]->(n2:Type2)` where `n1`, `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 (that is *simple* in the second form) going from a node of type `Type1` to a node of type `Type2`. All its arguments are individually optional, so the merest form of an expression is `()-[]->()` denoting a path made of two nodes connected by any edge. Conditions on attributes are expressed on node and edges variables in a `WHERE` clause.

Example 3: We denote by $\mathcal{P}$ the following fuzzy graph pattern:

```
1 (ar1:Article)-[part_of.series]->(s1),
2 (ar2:Article)-[part_of.series]->(s2),
3 (ar1)-[:creator]->(au1:Author),
4 (ar2)-[:creator]->(au1:Author),
5 (au1)-[(contributor+)|Length IS short]->(au2:Author)
6 WHERE
7 s1.id=WWW, s2.id=Pods,
8 ar2.year IS recent.
```

The graph of Fig. 5 is a graphical representation of pattern $\mathcal{P}$ where the dashed edge denotes a path and information in italics denotes a node type or an additional condition on node or edge attributes. This pattern "models" information concerning authors (`au2`) who have, among their close contributors, an author (`au1`) who published a paper (`ar1`) in `WWW` and also published a paper (`ar2`) in `Pods` recently (`ar2.year IS recent`). ◇

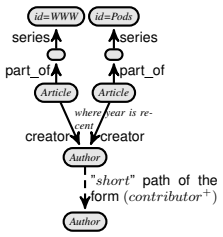

Figure 5. Pattern $\mathcal{P}$

We now give the definition of fuzzy graph pattern matching.

Definition 5 (Fuzzy graph pattern matching): A (fuzzy) data graph $G = (V, R, \kappa, \zeta)$ *matches* a fuzzy graph pattern $\mathcal{P} = (V_{\mathcal{P}}, E_{\mathcal{P}}, f_e^{path}, f_n^{cond}, f_e^{cond}, f_n^{type})$ with a satisfaction degree denoted by $\mu_{\mathcal{P}}(G)$ if there exists a binary relation $S \subseteq V_{\mathcal{P}} \times V$ representing an injective function from $V_{\mathcal{P}}$ to V s. t. (i) (mapping nodes) for each node $u \in V_{\mathcal{P}}$, there exists a node $v \in V$ s. t. $(u, v) \in S$; (ii) (mapping edges) for each edge $(u, u') \in E_{\mathcal{P}}$, there exist two nodes v and v' of V s. t. $\{(u, v), (u', v')\} \subseteq S$ and there is a path p in G from v to v' s. t. p matches $f_e^{path}(u, u')$ (recall that in case of matching, a satisfaction degree is associated, cf. Definition 3); (iii) (checking conditions on node attributes and type) for each pair $(u, v) \in S$, $\kappa(v) \vdash f_n^{cond}(u)$ (the semantics of $\vdash$ is clear from the context here) and $f_n^{type}(u) = Type(v)$ and (iv) (checking conditions on edge attributes) the same reasoning is trivially applied to conditions on attributes for edges labeled with a simple fuzzy regular expression in $E_{\mathcal{P}}$, that is to say $\zeta(v, v') \vdash f_e^{cond}(u, u')$.

The value of $\mu_{\mathcal{P}}(G)$ is the minimum of the satisfaction degrees produced by the mappings and conditions from (ii), (iii) and (iv). If there is no relation S satisfying the previous conditions, then $\mu_{\mathcal{P}}(G) = 0$, i.e., G does not match $\mathcal{P}$.

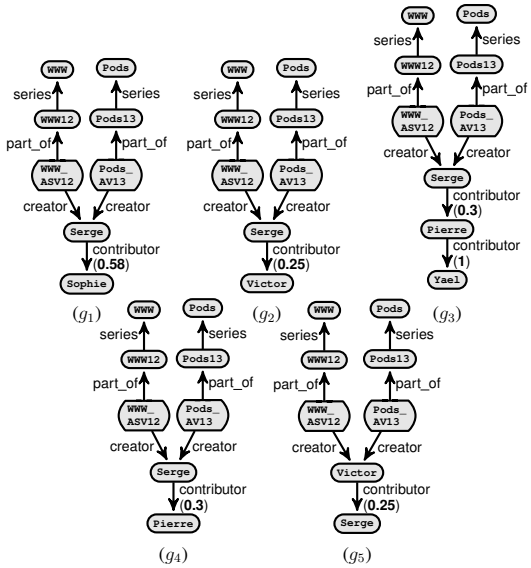

Figure 6. Subgraphs of $\mathcal{DB}$ matching $\mathcal{P}$

Example 4: Fig. 6 gives the set of subgraphs of $\mathcal{DB}$ matching the pattern $\mathcal{P}$ of Example 3. `au1` is necessarily either `Serge` or `Victor` who are the only authors of $\mathcal{DB}$ that wrote a paper at `WWW` and a recent paper in `Pods`. If we suppose that $\mu_{recent}(2013) = 0.75$ and that p is the path going from $au1$ to $au2$, then, as the satisfaction degree is the minimum of satisfaction degrees induced by lines 5, we have $\mu_{\mathcal{P}}(g_1) = 0.75$ (as $\mu_{short}(Length(p)) = \mu_{short}(1.72) = 1$), $\mu_{\mathcal{P}}(g_2) = 0.5$ (as $\mu_{short}(Length(p)) = \mu_{short}(4) = 0.5$), $\mu_{\mathcal{P}}(g_3) = 0.33$ (as $\mu_{short}(Length(p)) = \mu_{short}(4.33) = 0.33$), $\mu_{\mathcal{P}}(g_4) = 0.75$ (as $\mu_{short}(Length(p)) = \mu_{short}(3.33) = 0.83$) and $\mu_{\mathcal{P}}(g_5) = 0.5$ (as $\mu_{short}(Length(p)) = \mu_{short}(4) = 0.5$).

Let us now move to the definition of the fuzzy operators and expressions of the algebra. Even if the graph database contains a single graph, a query may return a set of graphs as several subgraphs may match a pattern as shown in Example 3. Graphs of a set do not necessarily have the same structure. A satisfaction degree is associated with each graph. A set of pairs $\langle\, graph\,, degree\,\rangle$ is nothing but a fuzzy set of graphs. Hence, each operator of the algebra takes one or more (depending on the arity of the operation) fuzzy set(s) of graphs as an input and generates a fuzzy

set of graphs as an output. All of the operators of the algebra operate in closed form. Applying an operation to the whole initial database means applying the operation to the singleton $\{\langle \mathcal{DB}, 1 \rangle\}$. *Expressions of the algebra* are defined inductively as usual: (i) a (fuzzy) graph database $\mathcal{DB}$ is an expression of the algebra, and (ii) if $e_1, \cdots, e_n$ are expressions and O is an operator of arity n then $O(e_1, \cdots, e_n)$ is an expression of the algebra.

Definition 6 (Selection operator): The selection operator σ takes as an input a fuzzy graph pattern $\mathcal{P}$ and a fuzzy set $\mathcal{G}$ of graphs. It returns a fuzzy set composed of all subgraphs of $\mathcal{G}$ that match the fuzzy graph pattern.

$$\sigma_{\mathcal{P}}(\mathcal{G}) = \{\langle s, \min(d, \mu_{\mathcal{P}}(s)) \rangle \mid \mu_{\mathcal{P}}(s) > 0\}$$

where s is a subgraph of g s.t. $\langle g, d \rangle \in \mathcal{G}$. In case of duplicates (a same graph appearing with several satisfaction degrees), the highest satisfaction degree is kept.$\diamond$

Example 5: Fig. 6 is the answer of $\sigma_{\mathcal{P}}(\mathcal{DB})$ where $\mathcal{P}$ is the pattern of Example 3 and $\mathcal{DB}$ is the database of Fig. 2.

Definition 7 (Alpha-cut operator): This operation performs an α-cut on a fuzzy relation of a graph (cf. section III-2). The alpha-cut operator Cut takes as input a fuzzy set of graphs $\mathcal{G}$, a label $e \in E$ and a degree $\alpha \in [0^+, 1]$. It produces a fuzzy set of graphs defined as follows:

$$Cut_{e,\alpha}(\mathcal{G}) = \{\langle (V, R', \kappa, \zeta), d \rangle \mid \langle (V, R, \kappa, \zeta), d \rangle \in \mathcal{G}\}$$

where $R' = \{\rho_l \mid \rho_l \in R \text{ and } l \neq e\} \cup \{\rho_e^{\alpha}\}$.$\diamond$
Note that the alpha-cut is seen here as an operation that "updates" the ρ_e relation by performing an α-cut on it, and so makes ρ_e crisp instead of fuzzy. It keeps the data embedded in edges labeled by e.

Definition 8 (Projection operators): The projection operation "on edges", Π^{edges}, removes relations from a graph. The projection operator Π^{edges} takes as input a fuzzy set of graphs $\mathcal{G}$, a set of labels $\mathcal{L} \subseteq E$. It returns a fuzzy set of graphs defined as follows:

$$\Pi^{edges}_{\mathcal{L}}(\mathcal{G}) = \{\langle (V, R', \kappa, \zeta), d \rangle \mid \langle (V, R, \kappa, \zeta), d \rangle \in \mathcal{G}\}$$

where $R' = \{\rho_e \mid \rho_e \in R \text{ and } e \in \mathcal{L}\}$.

The projection operation "on nodes", Π^{nodes}, removes nodes from a graph. The projection operator Π^{nodes} takes as input a fuzzy set of graphs $\mathcal{G}$, a set of node types $\mathcal{T}$. It produces a fuzzy set of graphs defined as follows:

$$\Pi^{nodes}_{\mathcal{T}}(\mathcal{G}) = \{\langle (V', R', \kappa, \zeta), d \rangle \mid \langle (V, R, \kappa, \zeta), d \rangle \in \mathcal{G}\}$$

where $V' = \{v \mid v \in V \text{ and } Type(v) \in \mathcal{T}\}$ and R' is the restriction of R over $V' \times V'$.

For these operators, in case of duplicates (a same graph appearing with several satisfaction degrees), the highest satisfaction degree is kept.$\diamond$

Definition 9 (Set operators): Union, intersection and difference are defined according to fuzzy set classical operations (see Section II-B for references).

The authors of [5] also define operators for combining, merging crisp graphs and restructuring the answer obtained by the query, that could complete the algebra, but this is left for a future extension.

V. Towards a flexible query language

The algebra constitutes the basis of a query language called FUDGE implementing the selection operator, in which fuzzy preferences appear. A FUDGE query is composed of:

1) a list of `DEFINE` clauses for fuzzy term declaration. If a fuzzy term `fterm` corresponds to a trapezoidal function of the general form of Fig. 1 with the four positions (abscissa) `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 like the term *short* of Fig. 3 then the clause has the form `DEFINEDESC fterm AS` (γ, δ) (there is the corresponding `DEFINEASC` clause for increasing functions).
2) a `MATCH` clause of the form `MATCH pattern WHERE conditions` where `pattern` is a fuzzy graph motif and `conditions` is the set of conditions attached to the pattern.

A FUDGE query example follows:

```
1  DEFINEDESC short AS (3,5)
2  DEFINEASC recent AS (2010,2014)
3  IN
4  MATCH
5  (ar1:Article)-[part_of.series]->(s1),
6  (ar2:Article)-[part_of.series]->(s2),
7  (ar1)-[:creator]->(au1:Author),
8  (ar2)-[:creator]->(au1:Author),
9  (au1)-[(contributor+)|Length IS short]->(au2:Author)
10 WHERE
11 s1.id=WWW AND s2.id=Pods AND ar2.year IS recent
```

In this example, the `DEFINEDESC` clause of line 1 defines the fuzzy term *short* of Fig. 3, and the following clause defines another fuzzy term *recent*. The pattern in the `MATCH`/`WHERE` clause is the pattern of Example 3.

As a proof-of-concept of our approach, we developed a software prototype for evaluating FUDGE queries. The FUDGE prototype, downloadable at `www-shaman.irisa.fr/fudge-prototype`, is based on the Neo4j system [15] that implements the CYPHER (crisp) query language. We extended then interactive Neo4j REPL Console Rabbithole [26]. We decided to use a crisp graph database management system for implementing the FUDGE query evaluator. This allows not only to access all the functionalities offered by the query language of the crisp engine, but also not to have to implement a whole query evaluator engine.

Then two problems arise. The first problem is to model a fuzzy graph database into a crisp graph database. This can be done in the crisp *property graphs model*. In a crisp property graph, a set of properties, where a property is a key-value pair, can be bound to a node or an edge. Properties usually denote embedded data and meta-data for nodes, and properties of the relation for edges. A relation $\rho_e(x, y)$ of a

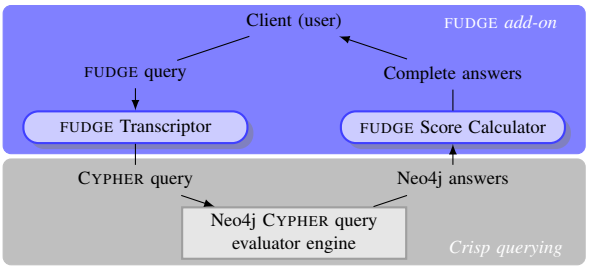

Figure 7. FUDGE query evaluation: prototype architecture

fuzzy graph database can be represented in a crisp property graph by considering the label of the edge connecting nodes x and y as being a pair (r, v) where v is the value of $\rho_e(x, y)$. Thus, the crisp property graph representation can simulate the fuzzy one by attaching to each edge of the property graph a supplementary property called *fdegree* carrying the degree value of the relations (supposing that *fdegree* now becomes a reserved keyword of the system). If needed, a similar mechanism allows to turn crisp nodes into fuzzy ones. The second problem is the evaluation of a FUDGE query that we dealt with by translating FUDGE queries into CYPHER crisp ones.

We implemented two modules: a module allowing the transcription of a FUDGE query into a (crisp) CYPHER one, which is then sent to the classical Neo4j query evaluation engine, and another module allowing the calculation of the satisfaction degree associated with each answer returned by the engine. Fig. 7 illustrates this architecture. Fig. 8 presents a screenshot of the software.

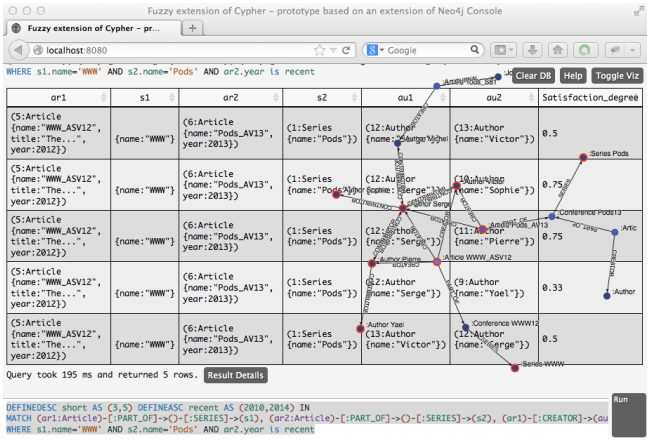

Figure 8. Screenshot of the FUDGE prototype

VI. RELATED WORK

As several models have been proposed to represent data having an implicit or explicit graph structure (see [12] for an overview of these models), literature includes a variety of query languages for graphs. Authors of [12], [27] and [11] propose complementary surveys of graph query languages defined in the past 25 years, including languages for querying graph-based object databases, semi-structured data, social networks and semantic web data. Reference [11] focuses on theoretical query languages for graph databases, and emphasizes that graph database management systems still lack query languages with a clear syntax and semantics. Our work goes towards filling this gap.

Functionalities that should be offered by a language for querying the topology of a crisp graph database are exhibited in [28], [8], [7], [9], [29], [27]. We summarize these (non-exclusive) functionalities hereafter, focusing on selection statements of the DML part of the language. Given a graph data G, *Adjacency queries* test node adjacency eg. check whether two nodes are adjacent, list all neighbors of a node; Given a vertex, *Reachability queries* search for topologically related vertices in G, where vertices are reachable by a *fixed-length path*, a *regular simple path* or a *shortest path*; *Pattern matching queries* look for all subgraphs of G that are isomorphic to a given graph pattern; *Data queries* specify conditions on the data embedded in G. Our algebra for fuzzy graph database expresses some flexible adjacency, reachability (as a rooted path is a special case of graph pattern), pattern matching and data queries on fuzzy and crisp graph databases. As a satisfaction degree is attached to each answer, rank-ordering them is straightforward.

Concerning flexible querying, [21] discusses different types of fuzzy preference criteria that appear relevant in the context of graph databases, without getting into the detail of how to express them using a formal query language. There are three main approaches allowing a flexible querying of graph databases: (i) *keyword-based query* approaches that completely ignore the data schema (see eg. [30]), which lack expressiveness for most querying use cases [31]; (ii) approaches that, given a "crisp" query, propose *approximate answers*, for instance by the implementation of a query relaxation or a approximate matching mechanism (see eg. [32], [33] or [31]); (iii) approaches allowing the user to introduce flexibility when formulating the query. Our approach belongs to this latter family for which many contributions concern the flexible extension of XPath [34], [35], [36] for querying semi-structured data (data trees). Such navigational languages behave well for querying graph databases [37] but no flexible extension was proposed in this specific case.

A work somewhat close to ours is [38] where authors propose a flexible extension of SPARQL allowing to introduce fuzzy terms and relations into the query language. A proof-of-concept implementation is proposed. This work only considers crisp graph databases, through.

VII. CONCLUSION

In this paper, we define a notion of fuzzy graph database and a fuzzy algebra for querying fuzzy and crisp graph databases in a flexible way. This algebra, based on fuzzy set theory and the notion of a fuzzy graph, consists of a data model and a family of operators that make it possible to

express preferences on i) the data embedded in the graph and ii) the structure of the graph. We also introduce a language based on the algebra, with a proof-of-concept implementation of this language. This language could be extended with new features. Interesting ones based on ordering and counting capabilities concern the *distance*, and *indegree*, *outdegree* or *centrality* of nodes. The other operators of the algebra could also be implemented.

REFERENCES

[1] D. Dubois and H. Prade, "Using fuzzy sets in database systems: Why and how?" in *Proc. of FQAS*, 1996, pp. 89–103.

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

[3] P. Bosc and O. Pivert, "SQLf: a relational database language for fuzzy querying," *IEEE Trans. on Fuzzy Systems*, vol. 3, pp. 1–17, 1995.

[4] R. Giugno and D. Shasha, "Graphgrep: A fast and universal method for querying graphs," in *ICPR (2)*, 2002, pp. 112–115.

[5] H. He and A. K. Singh, "Graphs-at-a-time: query language and access methods for graph databases," in *Proc. of SIGMOD'08*, 2008, pp. 405–418.

[6] 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 *Proc. of WAIM'10 Workshops*, 2010, pp. 37–48.

[7] 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 *ACM Southeast Regional Conf.*, 2010, p. 42.

[8] R. Angles, "A comparison of current graph database models," in *Proc. of ICDE Workshops*, 2012, pp. 171–177.

[9] M. Ciglan, A. Averbuch, and L. Hluchý, "Benchmarking traversal operations over graph databases," in *Proc. of ICDE Workshops (ICDEW)*, 2012, pp. 186–189.

[10] S. Batra and C. Tyagi, "Comparative analysis of relational and graph databases," *Intl. Journal of Soft Computing and Engineering*, vol. 2, no. 2, 2012.

[11] P. Barceló Baeza, "Querying graph databases," in *Proc. of PODS*, 2013, pp. 175–188.

[12] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Comput. Surv.*, vol. 40, no. 1, pp. 1–39, 2008.

[13] "AllegroGraph web site," franz.com/agraph/allegrograph.

[14] "InfiniteGraph web site," www.objectivity.com/infinitegraph.

[15] "Neo4j web site," www.neo4j.org.

[16] "Sparksee web site," sparsity-technologies.com.

[17] L.A. Zadeh, "Fuzzy sets," *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.

[18] D. Dubois and H. Prade, *Fundamentals of fuzzy sets*, ser. The Handbooks of Fuzzy Sets. Kluwer Academic, 2000, vol. 7.

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

[20] J. N. Mordeson and P. S. Nair, *Fuzzy Graphs and Fuzzy Hypergraphs*, ser. Studies in Fuzziness and Soft Computing. Springer, 2000, vol. 46.

[21] R. Yager, "Social network database querying based on computing with words," in *Flexible Approaches in Data, Information and Knowledge Management*, ser. Studies in Computational Intelligence. Springer, 2013.

[22] P. Bhattacharya and F. Suraweera, "An algorithm to compute the supremum of max-min powers and a property of fuzzy graphs," *Pattern Recognition Letters*, vol. 12, no. 7, pp. 413–420, 1991.

[23] J. Fodor and R. Yager, "Fuzzy-set theoretic operators and quantifiers," in *The Handbooks of Fuzzy Sets Series, vol. 1: Fundamentals of Fuzzy Sets*. Kluwer Academic Publishers, 2000, pp. 125–193.

[24] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu, "Adding regular expressions to graph reachability and pattern queries," *Frontiers of Computer Science*, vol. 6, no. 3, pp. 313–338, 2012.

[25] Neo Technology, "The Neo4j Manual v2.0.0," 2013, part III.

[26] "RabbitHole," neo4j.com/blog/rabbithole-the-neo4j-repl-console/.

[27] P. T. Wood, "Query languages for graph databases," *SIGMOD Record*, vol. 41, no. 1, pp. 50–60, 2012.

[28] R. Angles and C. Gutiérrez, "Querying RDF Data from a Graph Database Perspective," in *Proc. of European Semantic Web Conf. (ESWC)*, 2005, pp. 346–360.

[29] R. Angles, A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey, "Benchmarking database systems for social network applications," in *Proc. of the Intl. Workshop on Graph Data Management Experiences and Systems*, 2013.

[30] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: Ranked keyword searches on graphs," in *Proc. of SIGMOD*, 2007, pp. 305–316.

[31] F. Mandreoli, R. Martoglia, G. Villani, and W. Penzo, "Flexible query answering on graph-modeled data," in *Proc. of EDBT*, 2009, pp. 216–227.

[32] Y. Kanza and Y. Sagiv, "Flexible queries over semistructured data," in *Proc. of PODS*, 2001, pp. 40–51.

[33] P. Buche, J. Dibie-Barthélemy, and G. Hignette, "Flexible Querying of Fuzzy RDF Annotations Using Fuzzy Conceptual Graphs," in *Proc. of ICCS*, 2008, pp. 133–146.

[34] E. Damiani, S. Marrara, and G. Pasi, "FuzzyXPath: Using Fuzzy Logic an IR Features to Approximately Query XML Documents," in *Foundations of Fuzzy Logic and Soft Computing*, ser. LNCS. Springer, 2007, pp. 199–208.

[35] A. Campi, E. Damiani, S. Guinea, S. Marrara, G. Pasi, and P. Spoletini, "A Fuzzy Extension of the XPath Query Language," *J. Intell. Inf. Syst.*, vol. 33, no. 3, pp. 285–305, 2009.

[36] J. M. Almendros-Jiménez, A. Luna, and G. Moreno, "A Flexible XPath-based Query Language Implemented with Fuzzy Logic Programming," in *Proc. of the RuleML*. Springer-Verlag, 2011, pp. 186–193.

[37] L. Libkin, W. Martens, and D. Vrgoč, "Querying Graph Databases with XPath," in *Proc. of ICDT*, 2013, pp. 129–140.

[38] J. Cheng, Z. M. Ma, and L. Yan, "f-SPARQL: A flexible extension of SPARQL," in *Proc. of DEXA*, 2010, pp. 487–494.