Summarizing Temporal NoSQL Graph Databases

Arnaud Castelltort and Anne Laurent

LIRMM - CNRS UMR 5506 - Université Montpellier 2 - France
{*firstname.lastname*}*@lirmm.fr*

Abstract

Graph databases are spreading over many fields and have been recently put on the foreground with the emergence of the so called NoSQL graph databases that allow to deal with huge volumes of data (*e.g.,* scientific and social networks). Such graphs often evolve over time. There are several ways to consider the summary of graphs. In this paper, we detail the specificities of temporal NoSQL graph databases. Our proposal is twofolded. First, we propose a definition of temporal NoSQL graph databases. Second, we propose an original method to summarize them by considering several types of linguistic summaries, namely *structure summaries*, *data structure summaries* and *fuzzy summaries*. These summaries are said to be *temporal* when they contain time information. We introduce the methods to extract them and discuss the results obtained by experimenting our approach on synthetic and real databases.

Key words: Linguistic Summaries, Graph Databases, NoSQL, Temporal Graphs, Fuzzy Graph Mining.

1 Introduction

Graphs are known to be efficient for representing data in many applications, from linguistics to chemistry and social networks. For instance, graphs allow to draw in a very intuitive manner the relationships among people and between these people and the organizations they belong to.

Graphs are recognized to play an important role within the pattern recognition field [1], thus being a key technology for retrieving relevant information, as for fraud detection [2] or in social/biological interactions. Relevant information can be retrived either by considering data mining methods or by running predefined queries [3].

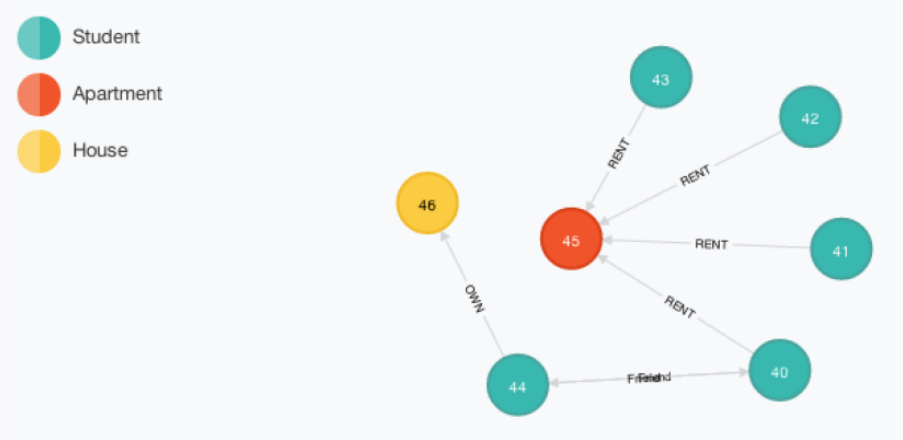


Fig. 1. Example of a Graph Database

Even if graphs are everywhere, their representation and usage have taken many forms in the literature and in real applications [4,5]. Theoretical works have been proposed in order to formally define the representations and treatments over such structures. More recently, they have been extensively used in the semantic web framework, especially with ontologies. However, with the emergence of big data, with more and more volouminous and complex data and treatments, it has been pointed out that robust database management systems were required and that classical relational database engines were not the solution, as shown in the performance comparisons done on this topic [6]. NoSQL are recognized for dealing with big data [7]. NoSQL graph databases have thus been proposed as the best alternative for managing huge volumes of graph data and complex queries over them. Several NoSQL engines exist, among which Neo4j is one of the most popular.

In the NoSQL graph database model, graph objects (nodes and relations) are dealt with. Complex information on both nodes and relations are managed as properties with $(key, value)$ pairs. On top of properties, nodes and relations can be stamped with *types*.

For instance, Fig. 1 displays the relations between people and the places they live in. The relation types depict whether they own or rent their housing. The form of housing can be apartment or house, this form being stored as the node type. Information on nodes (*e.g.,* age of people) and relations (*e.g.,* monthly rental fees) can be provided as node and relation properties (*e.g.,* $key = age$).

Operations can be defined for reading and writing data. NoSQL graph databases provide users with several ways to query, whether it be at the programmation level, via API, or though a declarative query language (Cypher for Neo4j). These operations can even be extended to fuzzy queries [8].

Many graphs contain time-varient information. For instance, in social networking, the relations between people and the implication of people within

organizations are timestamped over timelines. The links between animals and their ecosystem evolve depending on seasons and on the evolution of the environment over time.

Analysing such historial information is crucial for understanding such complex systems and dealing with the associated challenges. Graphs and databases have dealt with time evolution by proposing many frameworks, especially for graphs [9,10], for XML, RDF and web data [11–16], for relational databases [17–19]. We can distinguish between two main models for managing time-related information. The first model is based on sequences of snapshots of the data, also known as versions. The second model integrates time information within the database. Time information is mostly defined as time points or time intervals. Times series are also sometimes managed in dedicated database engines as done in the data streams management systems (DSMS) proposed some years ago [20].

In this paper, we consider the second model for temporal NoSQL graph databases. In our framework, NoSQL graphs can embed time information both in node values and relation values.

As the amount and complexity of information is growing, the need for summaries is increasing. The literature is also rich with propositions to deal with linguistic summaries of relational databases including or not temporal information and time series [21,22]. However, these works cannot be easily applied on temporal NoSQL graph databases. The summaries we aim at discovering must indeed be transposable into linguistic summaries. Moreover such databases combine several criteria that have never been dealt with altogether: temporal information, types of nodes and relations, complex information contained in the $(key, value)$ properties.

In this paper, we thus propose several types of linguistic summaries over temporal NoSQL graph databases. Experiments have been run over synthetic and real databases, showing the interest of our proposal.

The paper is organized as follows. Section 2 reports existing work on NoSQL graph databases focusing on time-varient data and on linguistic summarization. The section also proposes a definition of temporal NoSQL graph databases used as a foundation for the rest of the paper. Section 3 introduces our definitions for linguistic summaries of temporal NoSQL graph databases. Section 4 presents the quality measures proposed over the linguistic summaries in the specific context of temporal NoSQL graph databases. Section 5 introduces the queries and algorithms for extracting the summaries, by highlighting the power of NoSQL graph databases and demonstrating it on a synthetic database. Section 6 reports the experiments performed on a real dataset. Section 7 concludes

the paper and discusses some of the future works opened.

2 Background

Our work is strongly related to NoSQL graph databases and data summarization, we thus recall below the basics of these two topics.

2.1 Graphs

2.1.1 General Concepts

Graphs have been studied for a long time by mathematicians and computer scientists. A graph can be directed or not, labeled or not. It is defined as follows.

Def 1 (Graph) *A graph G is given by a pair (V, E) where V stands for a set of vertices and E stands for a set of edges with $E \subseteq (V \times V)$.*

Def 2 (Directed Graph) *A directed graph G is given by a pair (V, E) where V stands for a set of vertices and E stands for a set of edges with $E \subseteq \{V \times V\}$. That is E is a subset of all ordered permutations of V element pairs.*

When used in real world applications, graphs need to be provided with the capacity to label nodes and relations, thus leading to the so-called labeled graphs, or property graphs.

Def 3 (Labeled Directed Graph) *A labeled oriented graph G is given by a quadruplet (V, E, α, β) where*

- *α stands for the set of node labels;*
- *β stands for the set of edge labels;*
- *V stands for a set of vertices with $\forall v \in V, v = (id_v, \kappa_v)$ s.t. $\kappa_v \subseteq \alpha$ and id_v is the vertice identifier;*
- *E stands for a set of edges with $\forall e \in E, e = (id_e, (v_e^1, v_e^2), \lambda_e)$ s.t. $(v_e^1, v_e^2) \in \{V \times V\}$, $\lambda_e \subseteq \beta$ and id_e is the edge identifier.*

Graphs are used in many contexts. They have especially been used in the semantic web framework for ontologies. Several representation formats have been proposed, which have been completed with query languages such as SPARQL [23]. They have been extended for managing time-related information, as described below.

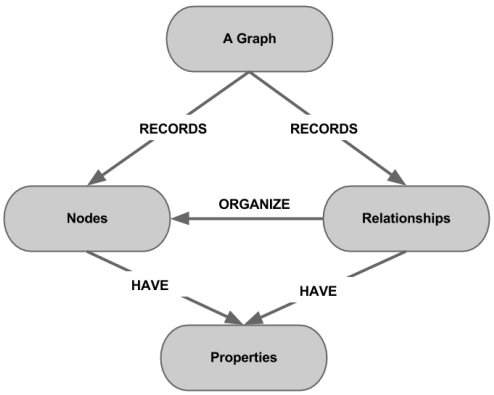


Fig. 2. Labeled Graph

2.1.2 Temporal Graphs

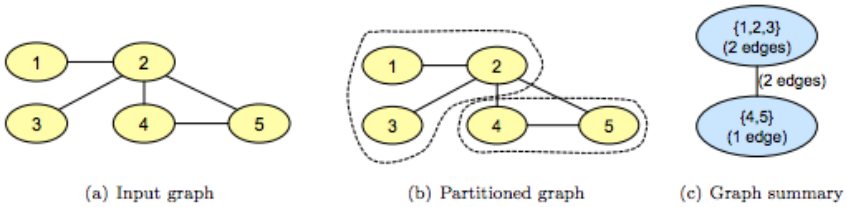
Graphs can describe time-varient information in several manners. Temporal information is crucial in many applications where graphs are useful. For instance, the evolution of the relations between elements, animals, people, etc. are studied by scientists, the evolution of the relations between customers and their favorite products is useful for marketers, etc.

Such temporal information is represented in different manners [24], as it has been the case for relational databases. Sequences of versions are often considered in order to capture the dynamics of the graph (as for instance in the Protege system for managing ontology changes [25]), for instance to analyse the evolution of communities in social networks and to detect social changes. Queries can be applied on such data. For this purpose, existing models have been extended, for instance for temporal RDF [26], XML and OWL data. Queries have also been extended, such as in the temporal SPARQL framework (T-SPARQL) [27], in the same way as temporal SQL has been introduced [28].

For summarizing, regarding works in databases, it has often been admitted that there are two main ways of managing time. The first manner is to consider versions of the data, taken as snapshots captured at some intervals of time. The second manner is to consider time labels associated with the data. When labeling, fixed time points are opposed to time intervals.

2.1.3 Graph Summarization

Data Summarization has been extensively studied in the last decades, aiming at producing information in the form of linguistic sentences, such as *Most of the students are young*. They have been especially formalized in the Yager's approach [29]. Such approaches consider relational data where source data are represented in the form of tuples defined over a schema. For instance,



(a) Input graph (b) Partitioned graph (c) Graph summary

Fig. 3. Graph Summarization using Supernodes [30]

the tuples $(John, 23, 45000)$, $(Mary, 32, 60000)$ and $(Bill, 38, 55000)$ are three tuples defined on the schema $(Name, Age, Salary)$.

Many works have dealt with these summaries, either considering how efficient algorithms can be designed, by studying relevant quality measures or by trying to extend them to other protoforms.

We do not recall here all the literature on fuzzy linguistic summaries which has been often presented, but we rather focus on the subject of graph data. In this paper, we consider such an approach which aims at extracting summaries corresponding to linguistic protoforms, such as Qy *are* P for supporting the above-mentioned summary *Most of the students are young*.

In this framework, two main characteristics have to be highlighted. First, graph databases are not provided with a strict and a priori given schema such as relational data. Regarding this, they are close to semi-structured data. Second, graph databases focus on relations.

Summarizing graph data has been considered for many years, aiming for instance at compressing such data with the use of supernodes as shown by Fig.3 from [30].

Graph summarization can be assimilated to graph mining in some extent. Graph (and tree) mining is seen as a problem of extracting *frequent* patterns (subgraphs/subtrees) from a large graph. It is often presented as an extension of the so-called itemset mining methods. Such methods have been successfully applied to large graphs by considering efficient approaches [31–33].

When considering evolving graphs, algorithms have for instance been proposed to follow the evolution of networks over time. One keypoint is to deal with the structure of the data. In this framework, some works in the literature have dealt with schema extraction in the context of semi-structured graph data, *i.e.,* XML data.

Semi-structured data is increasingly occurring since the advent of the Internet where full-text documents and databases are not the only forms of data any more and different applications need a medium for exchanging information.

In this paper, we address the benefit of automatically schema extraction and also its problems. Especially, this papers mainly talks about XML,the most representative semi-structured data.

The schema extraction problem is to identify a schema S from a given set of XML data documents D such that S captures the structural information of the documents in D in a minimal way. (*i.e.,* S is general and specific enough at the same time to cover D).

The schema extraction process is also referred to as *schema inference* [34]. The underlying structure of a given collection of XML documents can be described using Document Type Definitions (DTD), XML Schema or in a more general representation such as tree or graph. The structure extraction techniques in the literature target at inferring three kinds of representations: tree or graph summaries, DTD or XML Schema

2.2 NoSQL Graph Databases

NoSQL graph databases [5] are based on the above-presented concepts for the data they manage, with the following additional points:

- nodes and relations are provided with types;
- properties are defined over the nodes and relations being stored thanks to the $(key, value)$ paradigm which is very common in NoSQL databases.

It should be noted that types are distinguished from properties as done in NoSQL engines such as Neo4j [1] . These types appear in Fig. 1 as colors for nodes (*e.g.,* Student, House) and as labels for relationships (*e.g.,* Owns).

We propose below a formal definition of NoSQL graph databases generalizing the definition of labeled directed graphs.

Def 4 (NoSQL Graph Database) *A NoSQL graph database G is given by a tuple $(V, E, \theta, \tau, \alpha, \beta)$ where*

- *α stands for the set of node properties defined by (key:value) pairs;*
- *β stands for the set of edge properties defined by (key:value) pairs;*
- *θ stands for the set of node types;*
- *τ stands for the set of edge types;*
- *V stands for a set of vertices with $\forall v \in V, v = (id_v, t_v, \kappa_v)$ s.t. $t_v \subseteq \theta$ stands for the types of v, $\kappa_v \subseteq \alpha$ stands for the properties of v and id_v is the vertice*

[1] The term *type* is used for both nodes and relations in this paper, which is not the case in Neo4j that considers *types* on relationships and *labels* on nodes.

identifier;

- *E stands for a set of edges with $\forall e \in E, e = (id_e, (v_e^1, v_e^2), t_e, \lambda_e)$ s.t. $(v_e^1, v_e^2) \in \{V \times V\}$, $t_e \subseteq \tau$ stands for the types of e, $\lambda_e \subseteq \beta$ stands for the properties of e and id_e is the edge identifier.*

The set of properties of a node v is denoted by α_v, the set of types is denoted by θ_v. The set of properties of a relation e is denoted by β_e, the set of types is denoted by τ_e.

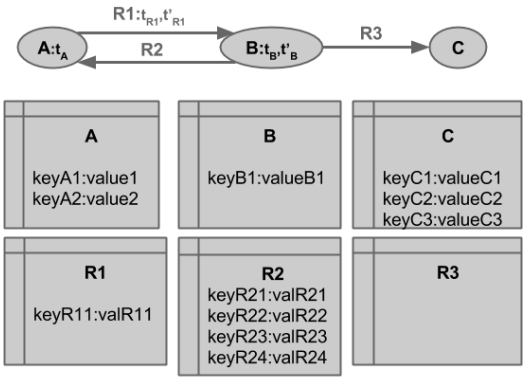


Fig. 4. Properties of Nodes and Relations in NoSQL Graph Databases

Fig. 4 shows a graph and its structure in $(key, values)$ pairs. On this figure, we have:

- $\alpha = \{(keyA1, valueA1), \ldots\}$
- $\beta = \{(keyR11, valueR11), \ldots\}$
- $V = \{(A, \{t_A\}, \{(keyA1, valueA1), (keyA2, valueA2)\}), (B, \{t_B, t'_B\}, \{(keyB1, valueB1)\}), (C, \emptyset, \{(keyC1, valueC1), (keyC2, valueC2), (keyC3, valueC3)\})$
- $E = \{(R1, (A, B), \{t_{R1}, t'_{R1}\}, \{(keyR11, valueR11)\}), \ldots\}$

Studies have shown that these technologies present good performances, much better than classical relational databases for representing and querying such large graph databases. Their exist several NoSQL graph database engines (OrientDB, Neo4J,HyperGraphDB, etc.) [4]. Neo4J is recognised as being one of the top ones regarding performance [35].

All NoSQL graph databases require the developers and users to use graph concepts to query data. As for any other repository, when querying such NoSQL graph databases, users either require specific focused knowledge (*e.g.,* retrieving Peter's friends) or ask for trend detection (*e.g.,* detecting trends and behaviours within social networks).

Queries are called *traversals*. A graph traversal refers to visiting elements, *i.e.* nodes and relations. There are three main ways to traverse a graph:

- programmaticaly, by the use of an API that helps developers to operate on the graph;
- by functional traversal, a traversal based on a sequence of functions applied to a graph;
- by declarative traversal, a way to explicit what we want to do and not how we want to do it. Then, the database engine defines the best way to achieve the goal.

In this paper, we focus on declarative queries over a NoSQL graph database. The Neo4j language is called Cypher.

For instance on Fig. 5, one query is displayed to return the customers who have visited the "Ritz" hotel. They are both displayed in the list and circled in red in the graph.

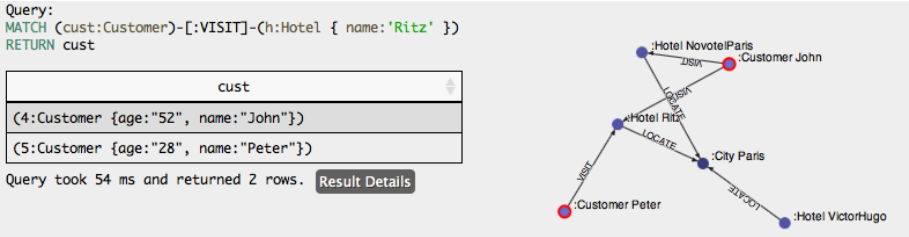


Fig. 5. Displaying the Result of a Cypher Query

Cypher clauses are similar to SQL ones. It is based on a "ASCII art" way of writing graph elements. For example, directed relations are written using the -[]->symbol. Types and labels are written after a semi-column (:).

More specifically, queries in Cypher have the following syntax [2] :

Listing 1. Query example on a Graph

```
1 [START]
2 [MATCH]
3 [OPTIONAL MATCH WHERE]
4 [WITH [ORDER BY] [SKIP] [LIMIT]]
5 RETURN [ORDER BY] [SKIP] [LIMIT]
```

As shown above, Cypher is comprised of several distinct clauses which are listed below in Listing 2.

[2] http://docs.neo4j.org/refcard/2.0/
http://docs.neo4j.org/chunked/milestone/cypher-query-lang.html

Listing 2. Cypher Clauses

```
START: Starting points in the graph, obtained via index lookups or by element IDs.
MATCH: The graph pattern to match, bound to the starting points in START.
WHERE: Filtering criteria.
RETURN: What to return.
CREATE: Creates nodes and relationships.
DELETE: Removes nodes, relationships and properties.
SET: Set values to properties.
FOREACH: Performs updating actions once per element in a list.
WITH: Divides a query into multiple, distinct parts.
```

Very few attempts have been made for managing time-related information in NoSQL graph databases, the only works being [36] and [37]. However these works focus on the management of the sequence of versions of the graph and do not consider the inclusion of time information such as timestamps and duration.

In this paper, we thus propose an original formal definition for temporal NoSQL graphs that are then used for extracting linguistic summaries. Temporal information is inherent to this definition. This definition is a generalization of the classical graphs defined above.

Def 5 (Temporal NoSQL Graph Database) *A temporal NoSQL graph G is given by a NoSQL graph database $(V, E, \theta, \tau, \alpha, \beta)$ where the following two properties hold:*

- $\forall\ v \in V, \exists!\ p \in \alpha_v$ *such that* $p.key = time$ *; and*
- $\forall\ e \in E,\ \exists!\ p \in \beta_e$ *such that* $p.key = time$.

It should be noted that temporal NoSQL graph databases generalize NoSQL graph database. The *time* property can indeed be set to a default value (*e.g.,* $CURRENT$) for every node and relation in the case that no temporal information is managed.

Fig. 6 shows such a temporal NoSQL graph database.

Some integrity constraints can be defined so that the timestamps of nodes and relations are compatible.

Example 1 *As a running example, we consider the graph database shown in Fig. 7 representing people and the place they live in.*

The following information is embedded in the graph:

- $V = \{(40, \{Student, Teacher\}, \{(Time : CURRENT), (Name : John), (Age : 32)\}), \ldots\}$;
- $E = \{\ldots, (R4, (40, 45), \{Rents\}, \{(Time : [2000, 2013])\}), \ldots\}$.

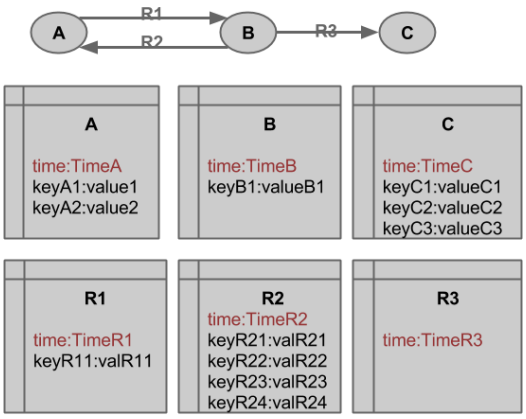


Fig. 6. Properties of Nodes and Relations with Time Management

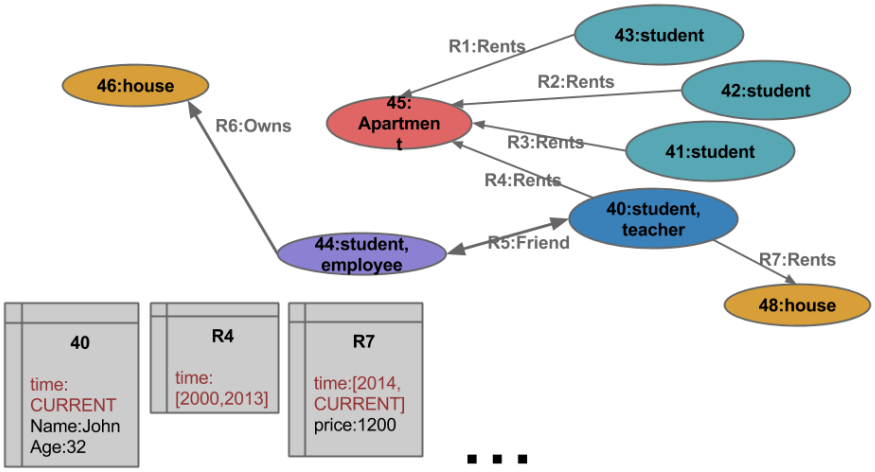


Fig. 7. Running Example: NoSQL Graph Database with Time Management

It should be noted that we have chosen to define time information at the granularity of nodes and relationships, which should be extended to a solution where timestamps are considered at the levels of types and keys. However, we argue that the concept of timestamp definition is rather a problem of database modeling than of data summarization. Moreover, this does not deeply change the definitions proposed below for data summarization while contributing to make it more understandable.

The rest of the paper is based on our proposal for temporal graphs and is described below by first providing the corresponding definitions and then by discussing the associated quality measures, by introducing the methods for extracting the summaries, before reporting the experimental results.

3 Building Temporal NoSQL Graph Summaries

As shown below, classical linguistic summaries cannot be directly applied on NoSQL graph databases, especially when considering temporal information. We thus propose below innovative definitions in such a context. Our proposition is based on the definition of protoforms accompagnied with quality measures and algorithms. This proposition has been assessed through experiments presented further in this paper.

3.1 Structure Summaries

Structure summaries are meant to retrieve the structure of the graph embedded in the element types, which could somehow be associated with the schema in relational databases. Such summaries could thus be linked with schema mining from the literature.

Def 6 (Structure Summary) *Let $G = (V, E, \theta, \tau, \alpha, \beta)$ be a temporal NoSQL graph database. A structure summary S of G is defined as $S = (a -[r]-> b, Q)$ where $a, b \in \theta$ (node types), $r \in \tau$ (relation type) and Q is a fuzzy quantifier.*

The structure summary can be expressed in a linguistic form as follows: In G, Q of the a r b

Example 2 *In the toy example, $(Student -[rent]-> apartment, Most)$, expressed as "*Most of the students rent an apartment*", is a structure summary.*

3.2 DataStructure Summaries

Data structure summaries are meant to refine structure summaries. They allow to distinguish some cases for the schema which depend on the value of some properties. For instance, it may be the case that employees rent rather apartments or houses depending on their salary.

Def 7 (Data Structure Summary) *Let $G = (V, E, \theta, \tau, \alpha, \beta)$ be a temporal NoSQL graph database. A Data Structure Summary S is defined as $S = (a.X -[r.Z]-> b.Y, Q)$ with $a, b \in \theta$ (node types), $r \in \tau$ (relation type), $X, Y \subseteq \alpha$ (node properties), $Z \subseteq \beta$ (relation properties) and Q a fuzzy quantifier.*

Example 3 *In the toy example, $Student(Age : 28) -[rent(fees : 1200)]-> apartment, Few$ is a data structure summary.*

Such summaries are extended in order to allow fuzzy linguistic labels in the refinement. It would indeed be both difficult and useless to define summaries on single values such as "the age is 28", as studied in fuzzy data mining for fuzzy association rule mining. By this way, it is possible to retrieve fuzzy linguistic summaries where *young* students and *low* rental fees are considered.

Def 8 (Fuzzy Data Structure Summary) *Let $G = (V, E, \theta, \tau, \alpha, \beta)$ be a temporal NoSQL graph database. Let F_α and F_β be sets of fuzzy properties. A Fuzzy Data Structure Summary S is defined as $S = (a.X$ –[$r.Z$]–>$b.Y, Q)$ with $a, b \in \theta$ (node types), $r \in \tau$ (relation type), $X, Y \subseteq \alpha \bigcup \mathcal{F}_\alpha$ (node properties and node fuzzy properties), $Z \subseteq \beta \bigcup F_\beta$ (relation properties and relation fuzzy properties) and Q a fuzzy quantifier.*

Example 4 *In the toy example, $Student(Age : young)$ –[$rent(fees : low)$]–> $apartment, Most$ is a fuzzy structure summary.*

3.3 Temporal Data Structure Summaries

A (fuzzy) data structure summary S is said to be temporal if it contains at leat one *time* property.

Def 9 (Temporal (Fuzzy) Data Structure Summary) *Let G be a temporal NoSQL graph database. A summary $S = (a.X$ –[$r.Z$]–>$b.Y, Q)$ of G is said to be temporal if and only if $\exists p \in X \bigcup Y \bigcup Z$ such that $p.key = time$.*

Example 5 *In the toy example, "$Student$–[$rent(time$:$[2010, 2014])$]–>$apartment, Most$" is a temporal data structure summary.*

Fuzzy temporal information is dealt with as shown in the literature, for instance by means of fuzzy intervals [38].

4 Quality Measures

Quality measures are crucial when working on summaries and patterns. They indeed provide information to the end-users about the veracity and quality of the proposed information. Many works have dealt with this topic, for instance in the field of association rule mining [39].

Regarding linguistic summaries, the most used measure is the degree of truth. The degree of truthdetermines the degree to which a linguistically quantified proposition is true. Truth values for simple protoforms above are defined by:

$$T(Qy'sareP) = \mu_Q\left(\frac{1}{n}\sum_{i=1}^{n}\mu_P(y_i)\right)$$

Where n is the number of objects (y_i) that are summarized, and μ_P , and μ_Q are the membership functions of the summarizer and quantifier, respectively.

In our framework, we propose several quality measures:

- Representativity
- Degree of truth
- Diversity

Their detailed definitions are given below. For the sake of simplicity, these definitions are provided with a general graph summary protoform denoted by $S = a -[r]-> b, Q$. In this protoform, a, b and r are meant to be extended to $a.X, b.Y, r.Z$ in the case of (fuzzy) data structure summaries.

We insist on the fact that the data we are dealing with are not the same as in the literature of linguistic summaries, even if some concepts can be taken. These differences are especially important for counting.

4.1 Counting in Temporal NoSQL Graph Databases

Counting is a key point in data mining and data summarization. It has for instance been discussed in [40] for fuzzy itemset mining, in [21] for sequential pattern mining and in [41] for tree mining.

In our context, two points have to be highlighted:

- all keys do not appear in all objects;
- temporal information have to be taken into account.

The first point comes from the fact that NoSQL graph databases correspond to the so-called semi-structured data. There is no strict schema followed by every node and relation. This is thus the case that types can appear in some nodes and relations in a free manner. In the same way, the same *keys* from (*key:value*) properties do not appear in every node and relation even if they share the same types. For instance, the age of the students do not always occur in the (*key:value*) pairs in the running example. Such a context could be dealt with as a null value problem. There have been many works on this topic in databases [42,43] and in data mining [44,45]. Proposals can be distinguished depending on the fact that they replace or ignore missing data.

In our framework, we argue that it does not make sense to replace these data as it is not that value but the whole key and value pair that is missing.

These principles are considered below in the *count* function.

4.2 Representativity

Representativity aims at informing the user about the presence of the pattern appearing in the summary in the databases. It could indeed be the case that most of the students rent an apartment but that students and apartments appear a very few times in a huge database.

For this purpose, we propose to display the ratio of occurrences of the relation being displayed over the number of relations in the graph.

Def 10 (Representativity) *Given a graph database G with set of edges E and a summary S, the representativity of S in G is defined as*

$$Representativity(S) = \frac{count(distinct(S))}{|E|}$$

Example 6 *In the toy example from Fig. 7, the representativity of the summary "$S = Student$ –[$rent$]–>$apartment, Most$" is given by the ratio between the number of times a relationship appears between a Student and an Apartment (s)he rents over the total number of relations. In this example, we thus have $Representatitvity(S) = \frac{4}{7} = 0.57$*

Roughly speaking, representativity can be compared to the *support* in association rule mining. It should be noted that the representativity does not depend on the quantifier.

4.3 Degree of Truth

The degree of truth determines the extent to which the relation appearing in the summary is veridic regarding the fuzzy quantifier. For instance, if the summary mentions that *most of the students rent an apartment,* then the degree of truth describes to which extend a high proportion of students rent an apartment.

There are two manners of counting such a degree. It could indeed be computed as the proportion of students who rent an apartment among the students or as

the proportion of students who rent an apartment over the number of students who rent their housing. These two definitions are provided below.

Def 11 (Degree of Truth) *Given a graph database G and a summary $S = a -[r]\text{->} b, Q$, the degrees of Truth of S in G are defined as*

$$Truth_1(S) = \mu_Q\left(\frac{count(distinct(S))}{count(distinct(a))}\right)$$

$$Truth_2(S) = \mu_Q\left(\frac{count(distinct(S))}{count(distinct(a -[r]\text{-> } (?)))}\right)$$

The second type of degree of truth is also called the *diversity of target source* denoted by D_T later on in this paper.

Example 7 *In the toy example from Fig. 7, the degree of truth of the summary "$S = Student -[rent]\text{->} apartment, Most$" is given by the membership degree to the fuzzy quantifier*[3] *of the ratio between the number of times a relationship appears between a Student and an Apartment (s)he rents over the number of relations of type Rents starting from a Student node. In this example, we thus have $Truth_2(S) = \mu_{Most}\left(\frac{4}{5}\right) = \mu_{Most}(0.8)$*

The ratio appearing in the definition of the degree of truth can be compared to the *confidence* in association rule mining which acts as a conditional probability.

4.4 Diversity

The representativity is a good quality indicator for describing to which extend the summary appears in the database. However, it can be refined by considering two separate measures that can be merged in a weighted manner.

The first submeasure we propose here is the D_R criterion describing the diversity of the relation.

The second submeasure we propose here is the D_T criterion describing the diversity of the target node.

The weighted average of these two submeasures is called the diversity.

[3] We do not mention here the detailed membership function of the *Most* quantifier which can be defined in a very classical manner as done in the literature of fuzzy quantifiers and fuzzy summaries.

Def 12 (Diversity of Relation) *Let $S = a -[r]-> b$ be a summary. The diversity of the relation for the pair of nodes (a, b) in S is defined as :*

$$D_R = \frac{|a -[r]-> b|}{|a -[?]-> b|}$$

If D_R is small for a summary, $a -[r]-> b$ it means that the weight of r for (a, b) is low.

Example 8 *For example, for a summary $(Person) -[RENT]-> (Place)$ the following subpatterns can be considered:*

- $(Person : Student) -[RENT]-> (Place : apartment)$
- $(Person : Student) -[RENT]-> (Place : Home)$
- $(Person : CEO) -[RENT]-> (Place : Home)$

If $D_R((Person : Student) -[RENT]-> (Place : Home)) < 1\%$, then this relation may be declared as being insignificant for the summary.

The diversity of relation is completed by the diversity of the target node.

Def 13 (Diversity of the Target Node) *For a pattern $a -[r]-> b$, the diversity of the target b for a couple (a, r) is defined as :*

$$D_T = \frac{|a -[r]-> b|}{|a -[r]-> (?)|}$$

If D_T is high, for a target node, it means that the weight of b for (a, r) is high.

From these two criteria, the diversity criterion is computed as a weighted average, as defined below.

Def 14 (Diversity) *Given a summary $S = a -[r]-> b, Q$ in a graph G, the degree of Truth of S in G is defined as:*

$$Diversity(S) = (\gamma_r D_R + \gamma_t D_T)$$

where $\gamma_r + \gamma_t = 1$

Example 9 *On the running example, the target $House$ is used 100% of the time for the pair $(employee, Owns)$ in the graph database.*

The Diversity metric for $(Employee) -[Rents]-> (House)$ is calculated as follows:

$$Diversity(S) = \gamma_r \frac{|Employee\text{-}[Rents]\text{-}\!\!>House|}{|Employee\text{-}[?]\text{-}\!\!>House|} + \gamma_t \frac{|Employee\text{-}[Rents]\text{-}\!\!>House|}{|Employee\text{-}[Rents]\text{-}\!\!>(?)|}$$

If Diversity was only composed of the D_R criterion, then this information would probably have been declared as irrelevant. The D_T criterion is balancing D_R.

The coefficients γ_r and γ_t should be chosen carefully taking into account what is the most efficient with respect to the context.

5 Extracting the Summaries

We detail below how to extract the above-defined summaries from NoSQL graph databases. It should be noted that most of the treatments we propose can be performed in an effective manner by defining queries over the NoSQL graph database, such leading to a more declarative than procedural way to extract summaries. This property is based on the fact that NoSQL graph databases provide powerfull pattern matching features. This is a keypoint, as inductive relational databases [46] have tried to achieve such a goal while failing because of the lack of high-level features in SQL.

The structure summaries can be retrieved by considering Cypher queries such as:

```
1 MATCH (a)-[r]->(m)
2 RETURN DISTINCT labels(a), type(r), labels(m), count(r)
```

With this query, all the structures are retrieved together with the number of times they appear. In order to compute the quality measures, as for instance the representativity, the total number of relations from the database has to be retrieved:

```
1 MATCH (a)-[r]->(m) RETURN count(r)
```

This result has to be compared for every summary with the number of relations and with the total number of times the node a appears.

The query from Listing 3 extracts the summaries from a graph and calculates the representativity and degree of truth for every summary.

Listing 3. Retrieving Structure Summaries

```
MATCH () -[ rel ]-> ()
WITH count(rel) AS nbRel
MATCH (a) -[ r ]-> (b)
WITH DISTINCT labels(a) AS labelsA, type(r) AS typeR, labels(b) AS labelsC,↩
     toFloat(count(*)) AS countS, nbRel
MATCH (a1) -[ r2 ]-> (m)
WHERE labels(a1)= labelsA AND type(r2)= typeR
WITH DISTINCT labelsA, typeR, labelsC, countS, labels(a1) AS labelsA1, type↩
    (r2) AS typeR2, count(*) AS count2, nbRel
RETURN labelsA, typeR, labelsC,
       tofloat(countS)/ nbRel AS Representativity,
       tofloat(countS)/ count2 AS Truth,
       MuMost(countS/count2) as TruthMost,
       MuFew(countS/count2) as TruthFew,
       MuVeryFew(countS/count2) as TruthVeryFew
```

Below is an example of the results being output:

Summary	Rep.	$TruthMost$	$TruthFew$	$TruthVeryFew$
Student-Rents-Apartment	4/7	$\mu_{Most}(4/5)$	$\mu_{Few}(4/5)$	$\mu_{VeryFew}(4/5)$
Student-Friend-Student	2/7	$\mu_{Most}(2/5)$	$\mu_{Few}(2/5)$	$\mu_{VeryFew}(2/5)$
Student-Owns-House	1/7	$\mu_{Most}(1/5)$	$\mu_{Few}(1/5)$	$\mu_{VeryFew}(1/5)$

Where Rep. stands for Representativity, and μ_{Most}, μ_{Few} and $\mu_{VeryFew}$ stand respectively for the membership function of the fuzzy subsets $Most$, Few and $VeryFew$ defined on the $[0, 1]$ universe which are implemented as the $MuMost$, $MuFew$ and $MuVeryFew$ functions in Cypher.

6 Experimental Results

6.1 Data

The real database being used deals with Cineasts, Movies and Actors [4] shown in Fig.8. The sample database contains about 12,000 movies, 50,000 actors and has size about 12.3 Mb.

To extract summaries from this graph, methods are implemented in Neo4j with the Cypher query language and the Java programming language, following a strict protocol so as to avoid any noisy results.

[4] http://docs.spring.io/spring-data/data-graph/snapshot-site/reference/html/#tutorial

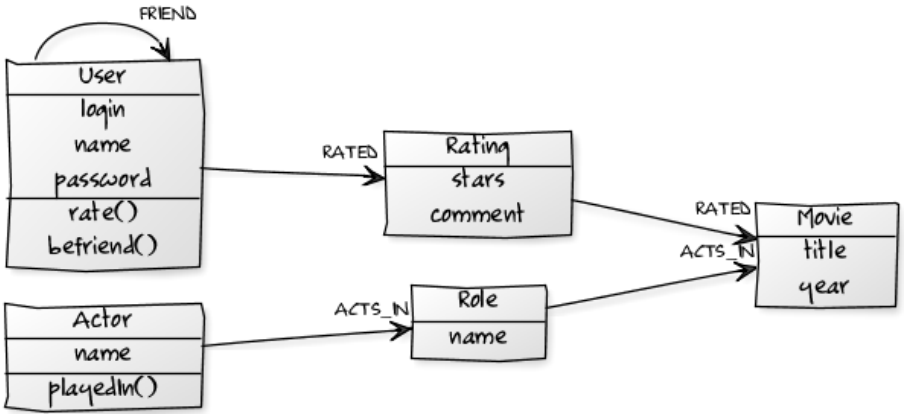


Fig. 8. Cineasts Movies and Actors sample database Model

Three fuzzy quantifiers have been defined: Most, Few and Very Few.

6.2 Protocol

The test environment and the methodology used are described below.

- Test Environment
  - Operating System: OS X 10.8.5 64 bits;
  - Processor: Intel Core i5 2.4 Ghz;
  - Java 1.7.0_45-b18 Java HotSpot(TM) Server VM (build 24.45-b08, mixed mode);
  - JVM configuration: Heap is set at 1024m, Perm at 256m
  - Database Engine: Neo4J 2.0.1
- Methodology
  - every test is run a series of 10 times to limit the side effects of the environment (I/O interruption, CPU usage by another process, etc.);
  - the outliers are deleted in order to avoid noisy results;
  - the environment (generated files, etc.) is cleaned before every test;
  - a new JVM is launched for every test to avoid side effect (like cache systems) and to avoid garbage collection run by the JVM.

It should be noted that the version of Java used is Oracle JVM and not OpenJDK that is not certified for Neo4j. Similarly, at the time of this report is written, Neo4j is not certified for running on Java 8.

6.3 Results

The Listing 4 displays some examples of structure summaries also reported on Fig. 6.3 for helping to retrieve the most relevant summaries while Listing 5

displays some examples of fuzzy data structure summaries.

Listing 4. Structure Summaries Extracted from the Movie Database

```
S1:(Person-[ACTS_IN]->Movie,Most); Truth(S1)=0.74
S2:(Person-[ACTS_IN]->Movie,Few); Truth(S2)=0.0
S3:(Person-[ACTS_IN]->Movie,VeryFew); Truth(S3)= 0.06
S4:(User-[FRIEND]->User,Most); Truth(S4)=0.0
S5:(User-[FRIEND]->User,Few); Truth(S5)=0.56
S6:(User-[FRIEND]->User,VeryFew); Truth(S6)=0.91
S7:(Person-[DIRECTED]->Movie,Most); Truth(S7)=0.0
S8:(Person-[DIRECTED]->Movie,Few); Truth(S8)=0.19
S9:(Person-[DIRECTED]->Movie,VeryFew); Truth(S9)=0.95
S10:(User-[RATED]->Movie,Most); Truth(S10)=0.72
S11:(User-[RATED]->Movie,Few); Truth(S11)=0.0
S12:(User-[RATED]->Movie,VeryFew); Truth(S12)=0.70
```

Listing 5. Fuzzy Data Structure Summaries Extracted from the Movie Database

```
DS1:(Person(young)-[ACTS_IN]->Movie,Most); Truth(DS1)=0.0
DS2:(Person(middleAge)-[ACTS_IN]->Movie,Most); Truth(DS2)=0.0
DS3:(Person(old)-[ACTS_IN]->Movie,Most); Truth(DS3)=1.0
DS4:(Person(young)-[ACTS_IN]->Movie,Few); Truth(DS4)=0.0
DS5:(Person(middleAge)-[ACTS_IN]->Movie,Few); Truth(DS5)=0.0
DS6:(Person(old)-[ACTS_IN]->Movie,Few); Truth(DS6)=0.0
DS7:(Person(young)-[ACTS_IN]->Movie,VeryFew); Truth(DS7)=0.24
DS8:(Person(middleAge)-[ACTS_IN]->Movie,VeryFew); Truth(DS8)=1.0
DS9:(Person(old)-[ACTS_IN]->Movie,VeryFew); Truth(DS9)=0.0
DS10:(Person(young)-[DIRECTED]->Movie,Most); Truth(DS10)=0.0
DS11:(Person(middleAge)-[DIRECTED]->Movie,Most); Truth(DS11)=0.0
DS12:(Person(old)-[DIRECTED]->Movie,Most); Truth(DS12)=1.0
DS13:(Person(young)-[DIRECTED]->Movie,Few); Truth(DS13)=0.0
DS14:(Person(middleAge)-[DIRECTED]->Movie,Few); Truth(DS14)=0.0
DS15:(Person(old)-[DIRECTED]->Movie,Few); Truth(DS15)=0.0
DS16:(Person(young)-[DIRECTED]->Movie,VeryFew); Truth(DS16)=0.21
DS17:(Person(middleAge)-[DIRECTED]->Movie,VeryFew); Truth(DS17)=0.47
DS18:(Person(old)-[DIRECTED]->Movie,VeryFew); Truth(DS18)=0.0
```

If, we consider the following summary extracted thanks to our method:

$$(Person - [ACTSIN] \rightarrow Movie, Most)$$

the degree of truth is computed as:

$$TruthMost(Person - [ACTSIN] \rightarrow Movie) = 0.74$$

This summary can be presented linguistically as *Most of Persons act in movies*, and this interpretation is true at the degree of 0.74, which makes it very believable.

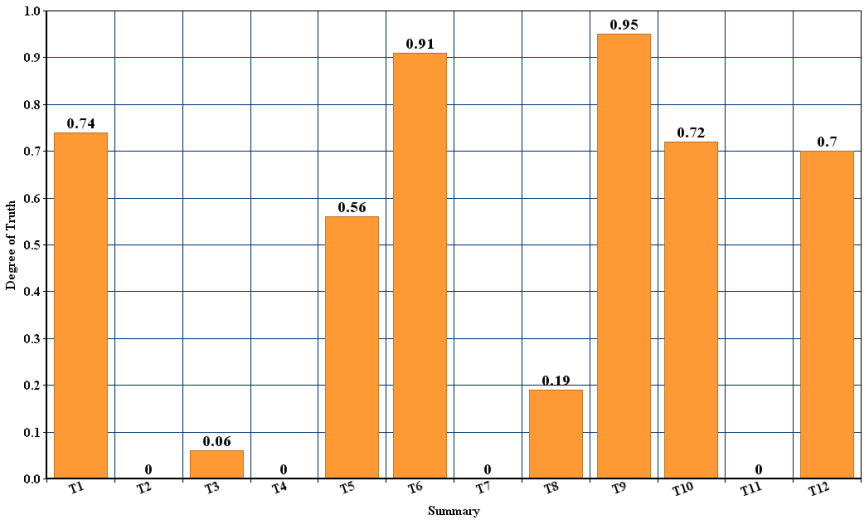


Fig. 9. Summaries and Degree

7 Conclusion and Perspectives

In this paper, we have considered an approach to summarize temporal NoSQL graph databases in the form of linguistic summaries. Such databases are quite specific with respect to the existing work as they combine several difficulties: focus on the relations, management of types for nodes and relationships, management of complex information in the (*key*:*value*) pairs, management of temporal information, etc.

The first contribution of this paper is to propose a formal definition for temporal NoSQL graph databases.

Several types of linguistic summaries are then proposed. They can be easily extracted using existing declarative query languages, making it possible to be deployed on every commercial tool. Experiments have been led on Neo4j using the Cypher query language, demonstrating the interest of our proposal.

Many perspectives are opened by this work. First, we aim at running more experiments in order to deal with scalability. The propositions could also be extended to several graphs. Moreover, we could like to integrate other types of summaries.

First, temporal information could be managed and summarized at several levels. In the above-presented work, the information is provided at the level of

nodes and relationships properties. It could also be provided at other levels:

- nodes and relationships types;
- (*key:value*) pairs in properties.

Moreover, spatio-temporal information could be considered as the spatial dimension is often interesting in real world problems. This is for instance the case in epidemiology or rumours circulating in social networks.

Another perspective is to consider extended linguistic summaries containing several relations. This work can be easily performed by keeping a declarative approach, the pattern matching features of NoSQL graph query languages being compatible with queries like:

Listing 6. Extended Summaries

```
1 MATCH (a)-[r1]->(m)-[r2]->n
2 RETURN DISTINCT labels(a), type(r1), labels(m), type(r2), labels(n)
```

However, such a perspective could be improved with data mining techniques in order to deal with the combinatory space of candidate summaries. The exploration of the space can thus be guided by APriori-like or Pattern-Growth-like algorithms.

The main focus are then:

- to study the monotonicity properties of the quality measures in order to allow data mining aglorithms to work (*e.g.,* representativity is anti-monotonic);
- to define aggregation measures in order to combine the relations and the quantifiers;
- to study how linguistic protoforms can be put on such extended summaries. Regargind the latter topic, it is indeed not easy to linguistically transcript summaries like:

  $$S = (students(age : young) \text{ --}[rent(fees : low)]\text{-->} Aparment \text{ --}[located]\text{-->} Suburbs \text{ --}[Classified(dangerous : low)]\text{-->} UrbanArea, Most).$$

  One way is to cut the summaries into several parts: *Most of the young students rent an apartment at a low price, this apartment being located in the low dangerous suburbs which are urban areas.* However, the automation of the process of the sentence building is not easy, all the more as several quantifiers can be combined: *Most of the students rent an apartment, but few of these apartments are located in the rich part of the city.*

As this latter form of summaries (*i.e.,* summaries containing contradictory quantifiers) can be very informative, we should first focus on this type of

summaries at level 2 (two relations, 2 quantifiers). Such summaries are very effective for pointing out exceptions and outliers.

Acknowledgements

The authors would like to thank Bui Dinh Duong for helping them to run experiments.

References

[1] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty Years Of Graph Matching In Pattern Recognition, International Journal of Pattern Recognition and Artificial Intelligence.

[2] G. Sadowski, P. Rathle, Fraud detection: Discovering connections with graph databases, in: White Paper - Neo Technology - Graphs are Everywhere, 2014.

[3] C. C. Aggarwal, H. Wang (Eds.), Managing and Mining Graph Data, Vol. 40 of Advances in Database Systems, Springer, 2010.

[4] R. Angles, C. Gutiérrez, Survey of graph database models, ACM Comput. Surv. 40 (1).

[5] I. Robinson, J. Webber, E. Eifrem, Graph Databases, O'Reilly, 2013.

[6] R. Cattell, Scalable SQL and NoSQL data stores, SIGMOD Record 39 (4) (2010) 12–27.

[7] J. Han, E. Haihong, G. Le, J. Du, Survey on nosql database, in: Proc. of the 6th International Conference on Pervasive Computing and Applications (ICPCA), 2011, pp. 363–366.

[8] A. Castelltort, A. Laurent, Fuzzy queries over nosql graph databases: Perspectives for extending the cypher language, in: International Conference on Processing and Management of Uncertainty in Knowledge-Based Systems, Springer, 2014.

[9] P. Holme, J. Saramaki, Temporal networks, Physics Reports 519 (3) (2012) 97–125.

[10] A. Anagnostopoulos, R. Kumar, M. Mahdian, E. Upfal, F. Vandin, Algorithms on evolving graphs, in: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference (ITCS), ACM, 2012, pp. 149–160.

[11] P. Plessers, O. D. Troyer, S. Casteleyn, Understanding ontology evolution: A change detection approach, J. Web Sem. 5 (1).

[12] J. Tappolet, A. Bernstein, Applied temporal RDF: Efcient temporal querying of RDF data with SPARQL, in: ESWC, 2009.

[13] J. Bao, L. Ding, D. L. Mcguinness, Semantic History: Towards Modeling and Publishing Changes of Online Semantic Data, in: CEUR, 2009.

[14] S. S. Chawathe, S. Abiteboul, J. Widom, Representing and querying changes in semistructured data, in: Proc. of the ICDE Conf., 1998.

[15] S. Auer, H. Herre, A versioning and evolution framework for rdf knowledge bases, in: Proceedings of the 6th international Andrei Ershov memorial conference on Perspectives of systems informatics, PSI'06, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 55–69.
URL http://dl.acm.org/citation.cfm?id=1760700.1760710

[16] V. Papavasileiou, G. Flouris, I. Fundulaki, D. Kotzinos, V. Christophides, High-level change detection in rdf(s) kbs, ACM Trans. Database Syst. 38 (1) (2013) 1:1–1:42.
URL http://doi.acm.org/http://dx.doi.org/10.1145/2445583.2445584

[17] S. Abiteboul, L. Herr, J. V. den Bussche, Temporal connectives versus explicit timestamps to query temporal databases, Journal of Computer and System Sciences 58 (1) (1999) 54 – 68.

[18] J. Chomicki, Temporal query languages: A survey, in: ICTL'94, 1994, pp. 506–534.

[19] R. T. Snodgrass, I. Ahn, A taxonomy of time in databases., in: S. B. Navathe (Ed.), SIGMOD Conference, ACM Press, 1985, pp. 236–246.

[20] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, J. Widom, STREAM: The Stanford Data Stream Management System, Tech. rep. (2004).

[21] C. Fiot, F. Masseglia, A. Laurent, M. Teisseire, TED and EVA: expressing temporal tendencies among quantitative variables using fuzzy sequential patterns, in: FUZZ-IEEE 2008, IEEE International Conference on Fuzzy Systems, Hong Kong, China, 1-6 June, 2008, Proceedings, IEEE, 2008, pp. 1861–1868.

[22] R. J. Almeida, M. Lesot, B. Bouchon-Meunier, U. Kaymak, G. Moyse, Linguistic summaries of categorical time series for septic shock patient data, in: FUZZ-IEEE 2013, IEEE International Conference on Fuzzy Systems, Hyderabad, India, 7-10 July, 2013, Proceedings., IEEE, 2013, pp. 1–8.
URL http://dx.doi.org/10.1109/FUZZ-IEEE.2013.6622581

[23] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF (January 2008).

[24] U. Khurana, An Introduction to Temporal Graph Data Management, Master's thesis, University of Maryland (2012).

[25] protege.stanford.edu, Protg (2012).
URL http://protege.stanford.edu/

[26] C. Gutierrez, C. Hurtado, R. Vaisman, Temporal rdf, in: In European Conference on the Semantic Web (ECSW05) (Best paper award, 2005, pp. 93–107.

[27] F. Grandi, T-sparql: a tsql2-like temporal query language for rdf, in: In International Workshop on on Querying Graph Structured Data, 2010, pp. 21–30.

[28] R. T. Snodgrass, Developing Time-oriented Database Applications in SQL, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

[29] R. R. Yager, A new approach to the summarization of data, Information Sciences 28 (1) (1982) 69 – 86.

[30] K. LeFevre, E. Terzi, Grass: Graph structure summarization, in: Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA, SIAM, 2010, pp. 454–465.

[31] D. J. Cook, L. B. Holder, Mining Graph Data, John Wiley & Sons, 2006.

[32] M. Kuramochi, G. Karypis, Frequent subgraph discovery, in: Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM '01, IEEE Computer Society, Washington, DC, USA, 2001, pp. 313–320.
URL http://dl.acm.org/citation.cfm?id=645496.658027

[33] X. Yan, J. Han, gspan: Graph-based substructure pattern mining, in: Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 721–.
URL http://dl.acm.org/citation.cfm?id=844380.844811

[34] G. J. Bex, F. Neven, S. Vansummeren, Inferring XML schema definitions from XML data, in: C. Koch, J. Gehrke, M. N. Garofalakis, D. Srivastava, K. Aberer, A. Deshpande, D. Florescu, C. Y. Chan, V. Ganti, C. Kanne, W. Klas, E. J. Neuhold (Eds.), Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007, ACM, 2007, pp. 998–1009.

[35] ThoughtWorks, Technology advisory board (May 2013).
URL
http://thoughtworks.fileburst.com/assets/technology-radar-may-2013.pdf

[36] U. K. and Amol Deshpande, Efficient snapshot retrieval over historical graph data, in: C. S. Jensen, C. M. Jermaine, X. Zhou (Eds.), ICDE, IEEE Computer Society, 2013, pp. 997–1008.

[37] A. Castelltort, A. Laurent, Representing history in graph-oriented nosql databases: A versioning system, in: I. S. Bajwa, M. A. Naeem, P. Pichappan (Eds.), Eighth International Conference on Digital Information Management (ICDIM 2013), Islamabad, Pakistan, September 10-12, 2013, IEEE, 2013, pp. 228–234.

[38] D. Dubois, H. Prade, Processing fuzzy temporal knowledge, IEEE Trans. on Systems, Man and Cybernetics 19 (4) (1989) 729–744.

[39] P.-N. Tan, V. Kumar, J. Srivastava, Selecting the right interestingness measure for association patterns, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '02, ACM, New York, NY, USA, 2002, pp. 32–41.

[40] D. Dubois, E. Hüllermeier, H. Prade, A systematic approach to the assessment of fuzzy association rules, Data Min. Knowl. Discov. 13 (2) (2006) 167–192.

[41] F. D. R. López, A. Laurent, P. Poncelet, M. Teisseire, Ftmnodes: Fuzzy tree mining based on partial inclusion, Fuzzy Sets and Systems 160 (15) (2009) 2224–2240.

[42] Y. Vassiliou, Null values in data base management a denotational semantics approach, in: Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data, SIGMOD '79, ACM, New York, NY, USA, 1979, pp. 162–169.
URL `http://doi.acm.org/10.1145/582095.582123`

[43] D. Maier, The Theory of Relational Databases, Computer Science Press, 1983.

[44] A. Ragel, B. Crémilleux, Treatment of missing values for association rules, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, 1998, pp. 258–270.

[45] A. Ragel, B. Crémilleux, MVC — a preprocessing method to deal with missing values, Knowledge-Based Systems Journal (1999) 285–291.

[46] L. De Raedt, A perspective on inductive databases, SIGKDD Explor. Newsl. 4 (2) (2002) 69–77.