Keywords-To-SPARQL Translation for RDF Data Search and Exploration

Katerina Gkirtzou[1] (✉), Kostis Karozos[2], Vasilis Vassalos[2], and Theodore Dalamagas[1]

[1] "Athena" Research Center, GR, Maroussi, Greece
{kgkirtzou,dalamag}@imis.athena-innovation.gr
[2] Athens University of Economics and Business, GR, Athens, Greece

Abstract. Linked Data is the most common practice for publishing and sharing information in the Data Web. As new data become available, their exploration is a fundamental step towards integration and interoperability. However, typical search methods as SPARQL queries require knowing both the SPARQL syntax and the vocabulary used in the data. For this reason, keyword-based search has been proposed, allowing an intuitive way for searching an RDF dataset. In this paper, we present a novel approach for keyword search on graph-structured data, and in particular temporal RDF graph, i.e. RDF data that involve temporal properties. Our method, instead of providing answers directly from the RDF data graph, automatically generates a set of candidate SPARQL queries that try to capture users information need as expressed by the keywords used. To support temporal exploration, our method is enriched with temporal operators allowing the user to explore data within predefined time ranges. To evaluate our approach, we perform an effectiveness study using two real-world datasets.

Keywords: Keyword search · Graph data · RDF · Linked Data

1 Introduction

More and more corporate, scientific, governmental and user-generated datasets break the walls of traditional "private" management within their production site, are published and become available for potential data consumers. The Linked Data (LD) is the most common practice for publishing, sharing and managing information in the Data Web, offering new ways of data integration and interoperability. The main concept in LD is that all resources published on the Web are uniquely identified by a *Uniform Resource Identifier* (URI), and typed links between URIs, also identified by URIs, are used to semantically connect resources[1]. LD is implemented with the RDF[2] technology: (*a*) RDF is used for the representation and modeling of structured and semi-structured data on the Web and (*b*) RDF links are used to interlink data from different data sources.

[1] http://www.w3.org/TR/ld-bp/.
[2] http://www.w3.org/RDF/.

S. Kapidakis et al. (Eds.): TPDL 2015, LNCS 9316, pp. 111–123, 2015.
DOI: 10.1007/978-3-319-24592-8_9

The RDF representation is a set of statements about resources. Such statements are known as triples. A triple is an expression of the form *subject predicate object*. The subject refers to a resource to be described, the predicate is usually a term from existing vocabularies, while the object can either be a literal (i.e., string) or another resource. A set of RDF triples can also be represented by a directed labelled graph, known as the RDF data graph. The most common way for searching an RDF data graph is the SPARQL query language [9]. SPARQL queries involve conjunctions and disjunctions of triple patterns which are matched on the RDF data graph. However, the use of SPARQL requires, apart from knowing its syntax, also the knowledge of the RDF schema used to model the data. For this reason, keyword-based search has been proposed in bibliography, allowing an intuitive way for searching an RDF dataset.

In this paper, we present a novel approach for keyword search on graph-structured data, and in particular RDF graph. Our method, instead of providing answers directly from the RDF data graph, generates automatically a set of candidate SPARQL queries that try to capture users information needs as expressed by the keywords used. Our approached is tailored to temporal RDF data, i.e. RDF data that involve temporal properties. To this end, our method is enriched with temporal operators allowing the user to explore data within predefined time ranges. To evaluate our approach, we perform an effectiveness study using two real-world datasets.

2 Problem Definition

We assume that the user wants to explore of a dataset using a *user query language* $\mathcal{Q}_U$, while the system supports queries only in a specified *system query language* $\mathcal{Q}_S$ different from $\mathcal{Q}_U$. To deal with this case, a transformation from $\mathcal{Q}_U$ to $\mathcal{Q}_S$ is required. We focus on datasets modeled in RDF graph form, defined as follows:

Definition 1. *An RDF data graph G is a tuple $(V, E, \mathcal{L})$ where:*

- *V is a finite set of vertices. V is defined as the disjoint union $V_E \cup V_C \cup V_V$, where V_E is a set of vertices representing RDF entities, V_C is a set of vertices representing RDF classes and V_V is a set of vertices representing literals.*
- *$E \subseteq V \times V$ is a finite set of order pairs (v_1, v_2), called edges, where $v_1, v_2 \in V$.*
- *$\mathcal{L} : \{V \cup E\} \rightarrow L$ is a function assigning $\forall v \in V$ and $\forall (v_1, v_2) \in E$ a label from alphabet L. The alphabet L is defined as the disjoint union $L_E \cup L_C \cup L_V \cup L_R \cup L_A \cup \{type, subclass\}$, where L_E is a set of labels for the RDF entities, L_C is a set of labels for the RDF classes, L_V is a set of labels for the literals, L_R is a set of labels for the inter-entities properties, L_A is a set of labels for the entity-to-attribute properties. The following restrictions apply:*
  - *$\mathcal{L}(v) \in L_E$ if and only if $v \in V_E$,*
  - *$\mathcal{L}(v) \in L_C$ if and only if $v \in V_C$,*
  - *$\mathcal{L}(v) \in L_V$ if and only if $v \in V_V$,*
  - *$\mathcal{L}((v_1, v_2)) \in L_R$ if and only if $v_1, v_2 \in V_E$,*
  - *$\mathcal{L}((v_1, v_2)) \in L_A$ if and only if $v_1 \in V_E$ and $v_2 \in V_V$,*

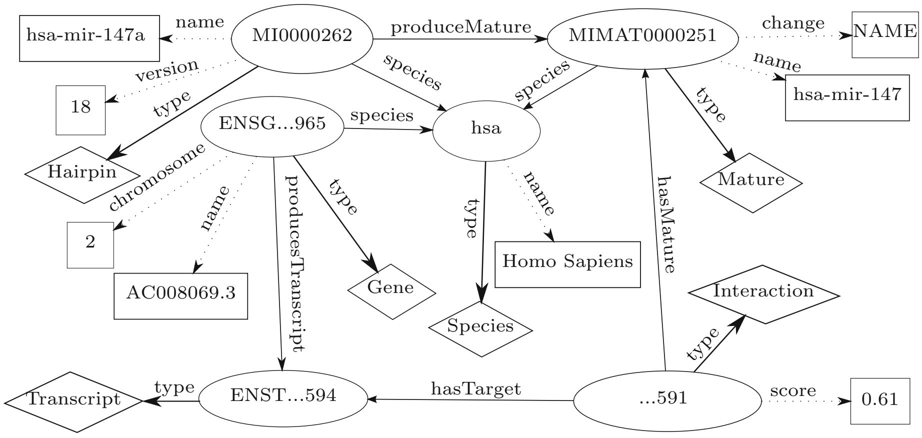

Fig. 1. An RDF subgraph from the DIANA dataset. If $v \in V_E$ then it has oval shape, if $v \in V_C$ a diamond shape and if $v \in V_V$ a rectangle shape. Also, dotted edges represent entity-to-attribute properties, while solid edges represent inter-entities properties.

- $\mathcal{L}((v_1, v_2)) = type$ *if and only if* $v_1 \in V_E$ *and* $v_2 \in V_C$*,*
- $\mathcal{L}((v_1, v_2)) = subclass$ *if and only if* $v_1, v_2 \in V_C$*.*

We consider two predefined types of edges, *type* and *subclass*, that have a special interpretation within RDF schema. The former states that an RDF entity is an instance of an RDF class, while the latter states class hierarchy [3]. The URIs of RDF entities define the set L_E, while the URIs of RDF class and RDF properties define the sets L_C and $L_R \cup L_A$, respectively. Finally, all literal data values define L_V. Figure 1 shows an RDF data subgraph for the DIANA dataset (see Sect. 6). The oval shape vertex `MI0000262` represents an RDF entity of the RDF class `Hairpin`. The entity's name is `hsa-mir-147a` and it is related with the entity `MIMAT0000251` of the RDF class `Mature` via the property `producesMature`.

In our scenario, the user query language $\mathcal{Q}_U$ is defined as a set of keywords $\mathcal{K} = \{k_1, k_2, \ldots, k_n\}$, while the system supports a query language $\mathcal{Q}_S$ that produces conjuctive SPARQL queries. These queries can also be viewed as graph patterns defined as follows:

Definition 2. *A graph pattern* G_Q *on an RDF data graph* $G = (V, E, \mathcal{L})$ *is a tuple* $(V_Q, E_Q, \mathcal{L}_Q)$ *where:*

- *V_Q is a finite set of vertices. V_Q is defined as the disjoint union $V_{VAR} \cup V_{CL} \cup V_{VAL}$, where V_{VAR} is a set of vertices representing variables, $V_{CL} \subseteq V_C$ is a set of vertices representing RDF classes and $V_{VAL} \subseteq V_V$ is a set of vertices representing literals.*
- *$E_Q \subseteq V_Q \times V_Q$ is a finite set of order pairs (v_1, v_2), where $v_1, v_2 \in V_Q$.*
- *$\mathcal{L}_Q : \{V_Q \cup E_Q\} \rightarrow L_Q$ is a function assigning $\forall v \in V_Q$ and $\forall (v_1, v_2) \in E_Q$ a label from alphabet L_Q. The alphabet L_Q is defined as the disjoint union $L_{VAR} \cup L_C \cup L_V \cup L_R \cup L_A \cup \{type\}$, where L_{VAR} is a set of labels for the variables. The following restrictions apply:*

- *$\mathcal{L}_Q(v) \in L_{VAR}$ if and only if $v \in V_{VAR}$,*
- *$\mathcal{L}_Q(v) \in L_C$ if and only if $v \in V_{CL}$,*
- *$\mathcal{L}_Q(v) \in L_V$ if and only if $v \in V_{VAL}$,*
- *$\mathcal{L}_Q((v_1, v_2)) \in L_R$ if and only if $v_1, v_2 \in V_{VAR}$,*
- *$\mathcal{L}_Q((v_1, v_2)) \in L_A$ if and only if $v_1 \in V_{VAR}$ and $v_2 \in \{V_{VAL} \cup V_{VAR}\}$,*
- *$\mathcal{L}_Q((v_1, v_2)) = type$ if and only if $v_1 \in V_{VAR}$ and $v_2 \in \{V_{CL} \cup V_{VAR}\}$.*

Given an RDF data graph G and a graph pattern G_Q, answers are constructed by mapping the variables $V_{VAR} \in G_Q$ to vertices $V \in G$ such that the substitution of variables in the graph pattern would yield a subgraph of G. Considering the above definitions, the problem is defined as follows:

Problem. *Given an RDF data graph G and a set of keywords $\mathcal{K} = \{k_1, k_2, \ldots, k_n\}$, we want to compute a ranked list of graph patterns G_Q given a ranking function R. The transformation of the keywords to graph pattern can be seen as a mapping fuction $\mu(\mathcal{K}) = G_Q$, where each keyword k_i will match to $L \setminus L_E$.*

3 Indexing RDF Graph Data

To assist the construction of graph patterns G_Q for a given set of keywords, we use two types of indices: (i) a term index and (ii) a schema-guide graph.

3.1 Term Index

A term index, given a keyword k_i, returns a set of matches from the RDF data graph G, $\mathcal{M}_i = \{m | m \in G = (V, E, \mathcal{L})\}$, along with other necessary information to assist query formation. More specifically, if $m \in L_A$, i.e. matches to a label of an entity-to-attribute property, then in the RDF data graph G exists at least one $e \in V_E, v \in V_V$ and $c \in V_C$ such that $(e, v) \in E$ with $\mathcal{L}((e, v)) = m$ and $(e, c) \in E$ with $\mathcal{L}((e, c)) = type$. In the index, we also keep the labels of the RDF classes c of the entities e that are subjects to the entity-to-attribute property with label m. Similarly, if $m \in L_R$, i.e. matches to a label of an inter-entities property, then in G exists at least one $e_s, e_o \in V_E, e_s \neq e_o$ and $c_s, c_o \in V_C$ such that $(e_s, e_o) \in E$ with $\mathcal{L}((e_s, e_o)) = m$, $(e_s, c_s) \in E$ with $\mathcal{L}((e_s, c_s)) = type$ and $(e_o, c_o) \in E$ with $\mathcal{L}((e_o, c_o)) = type$. In the index, we also keep the labels of the pairs of the RDF classes $\langle c_s, c_o \rangle$ of the entities e_s, e_o that are subjects and objects of the inter-entities property with label m, respectively. Finally, if $m \in V_V$, i.e. a literal, then in the RDF graph G exists at least one $e \in V_E, c \in V_C$ such that $a = (e, m) \in E$ and $(e, c) \in E$ with $\mathcal{L}((e, c)) = type$. It is possible that the same literal is met in multiple entities e of the same or even different classes c under both the same property or different properties a. To keep the index minimal, we keep one entry per property a for a given literal m, grouping only the labels of the RDF classes c. Table 1 shows analytically the information provided by the term index depending on the type of the matched element m.

Table 1. The information of the term index by the type of the matched element m.

Matched element	Term Index Structure	Notation
$m \in L_A$	$[m, \{\mathcal{L}(c_1), \ldots, \mathcal{L}(c_n)\}]$	$c_i \in V_C$ and $\mathcal{L}(c_i) \in L_C$
$m \in L_R$	$[m, \{\langle \mathcal{L}(c_{s_1}), \mathcal{L}(c_{o_1}) \rangle, \ldots, \langle \mathcal{L}(c_{s_n}), \mathcal{L}(c_{o_n}) \rangle\}]$	$c_{s_i}, c_{o_i} \in V_C$ and $\mathcal{L}(c_{s_i}), \mathcal{L}(c_{o_i}) \in L_C$
$m \in V_V$	$[m, \mathcal{L}(a), \{\mathcal{L}(c_1), \ldots, \mathcal{L}(c_n)\}]$	$\mathcal{L}(a) \in L_A, c_i \in V_C$ and $\mathcal{L}(c_i) \in L_C$

3.2 Schema-Guide Graph

The schema-guide graph is an aggregated representation of the RDF data graph. It is used to guide the query computation process and it is defined as follows:

Definition 3. *A schema-guide graph G_{SCM} of an RDF graph $G = (V, E, \mathcal{L})$ is a tuple $(V_{SCM}, E_{SCM}, \mathcal{L}_{SCM})$ where:*

- *V_{SCM} is a finite set of vertices defined as the disjoint union $V_C \cup \{Thing\}$,*
- *$E_{SCM} \subseteq V_{SCM} \times V_{SCM}$ is a finite set of order pairs (v_1, v_2), where $v_1, v_2 \in V_{SCM}$ and*
- *$\mathcal{L}_{SCM} : \{V_{SCM} \cup E_{SCM}\} \rightarrow L_{SCM}$ is a function assigning $\forall v \in V_{SCM}$ and $\forall e \in E_{SCM}$ a label from alphabet L_{SCM}, which is defined as the disjoint union $L_C \cup L_R \cup \{Thing, subclass\}$. The following restrictions apply:*
  - *$\mathcal{L}_{SCM}(v) \in L_C$ if and only if $v \in V_C$,*
  - *$\mathcal{L}_{SCM}(v) = Thing$ if and only if $v = Thing$,*
  - *$\mathcal{L}_{SCM}((v_1, v_2)) \in \{L_R, subclass\}$ if and only if $v_1, v_2 \in V_C$.*

Every vertex $c \in V_C$ of the schema-guide graph G_{SCM} represents all the entities vertices $e \in V_E \subset V$ of the RDF data graph G that have RDF type c and $\mathcal{L}_{SCM}(c) \in L_C$. On the other hand, vertex $Thing \in V_{SCM}$ with $\mathcal{L}_{SCM}(Thing) = Thing$ represents all entities vertices $e \in V_E \subset V$ of the RDF data graph G that have no given type. Similarly, an edge $(c_1, c_2) \in E_{SCM}$ of the schema-guide

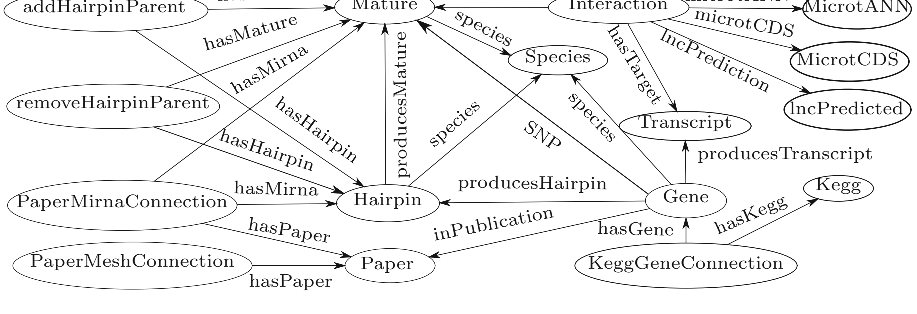

Fig. 2. The schema-guide graph of DIANA dataset.

graph G_{SCM} represents all edges $(e_1, e_2) \in E$ where $e_1, e_2 \in V_E$ of the data graph G if and only if $(e_1, c_1) \in E$ with $\mathcal{L}((e_1, c_1)) = \textit{type}$ and $(e_2, c_2) \in E$ with $\mathcal{L}((e_2, c_2)) = \textit{type}$. In this case, $\mathcal{L}_{SCM}((c_1, c_2) = \mathcal{L}((e_1, e_2)) \in L_R$. Finally, the edge $(c_1, c_2) \in E_{SCM}$ with $\mathcal{L}((c_2, c_2)) = \textit{subclass}$ represents class hierarchy.

Figure 2 shows an example of the schema-guide graph for the DIANA dataset. Note that vertices here represent all RDF entities from the RDF graph of a specific type rather than specific RDF entity. For example vertex `MI0000262` in Fig. 1 is represented by vertex `Hairpin` in Fig. 2. Similarly, the property `producesMature` between the RDF entities `MI0000262` and `MIMAT0000251` in Fig. 1 is represented by the abstract edge `producesMature` between vertex `Hairpin` and vertex `Mature` in Fig. 2.

4 Query Pattern Graph

To compute the graph patterns as a response to use keyword-based queries, we perform the following steps: (*1*) for each keyword $k_i \in \mathcal{K}$ we retrieve all its matches $\mathcal{M}_i$ on the RDF data graph, (*2*) we calculate all possible combinations of the matched elements $\mathcal{C} = \mathcal{M}_1 \times \ldots \times \mathcal{M}_n = \{c = (m_1, \ldots, m_n) | m_i \in \mathcal{M}_i, \forall i = 1, \ldots, n\}$, (*3*) for each combination $c = (m_1, \ldots, m_n) \in \mathcal{C}$ that contains one matched element m_i per keyword k_i, we create an augmented schema-guide graph G_{AUG}, (*4*) for each augmented schema-guide graph G_{AUG}, we generate the graph query pattern G_{QP} which is used to form a SPARQL query and (*5*) we rank each query pattern graph G_{QP} based on a ranking function $\mathcal{R}$.

4.1 Augmented Schema-Guide Graph

The augmented schema-guide graph is used as a data guide for the query pattern formation and it is defined as follows:

Definition 4. *An augmented schema-guide graph G_{AUG} of an RDF data graph $G = (V, E, \mathcal{L})$ is a tuple $(V_{AUG}, E_{AUG}, \mathcal{L}_{AUG})$ where:*

- *V_{AUG} is a finite set of vertices defined as the disjoint union $V_C \cup V_{VAL} \cup V_U \cup \{Thing\}$, where $V_{VAL} \subseteq V_V$ is a set of vertices representing literals and V_U is a set of vertices representing unknown literal values,*
- *$E_{AUG} \subseteq V_{AUG} \times V_{AUG}$ is a finite set of order pairs (v_1, v_2), where $v_1, v_2 \in V_{AUG}$ and*
- *$\mathcal{L}_{AUG} : \{V_{AUG} \cup E_{AUG}\} \rightarrow L_{AUG}$ is a function assigning $\forall v \in V_{AUG}$ and $\forall e \in E_{AUG}$ a label from an alphabet L_{AUG}, defined as the disjoint union $L_C \cup L_V \cup L_R \cup L_A \cup \{Thing, subclass\}$. The following restrictions apply:*
  - *$\mathcal{L}_{AUG}(v) \in L_C$ if and only if $v \in V_C$,*
  - *$\mathcal{L}_{AUG}(v) \in L_V$ if and only if $v \in V_{VAL}$,*
  - *$\mathcal{L}_{AUG}(v) = \emptyset$ if and only if $v \in V_U$,*
  - *$\mathcal{L}_{AUG}(v) = Thing$ if and only if $v = Thing$,*
  - *$\mathcal{L}_{AUG}((v_1, v_2)) \in \{L_R \cup subclass\}$ if and only if $v_1, v_2 \in V_C$,*
  - *$\mathcal{L}_{AUG}((v_1, v_2)) \in L_A$ if and only if $v_1 \in V_C$ and $v_2 \in \{V_{VAL} \cup V_U\}$.*

Given a set $(m_1, \ldots, m_n) \in \mathcal{C}$, we construct the augmented schema-guide graph G_{AUG} from the schema-guide graph G_{SCM} as follows:

- If $m_i \in L_V$, add an edge (c, v_{m_i}) and a vertex v_{m_i} with $\mathcal{L}_{AUG}(v_{m_i}) = m_i$. The newly inserted edge (c, m_i) will be attached to the proper $c \in V_C \cup$ *Thing* vertex and labelled $\mathcal{L}_{AUG}(c, v_{m_i}) = \mathcal{L}(a)$, according to the information provided by term index for m_i.
- If $m_i \in L_A$, add an edge (c, v) and a vertex v. The newly inserted edge (c, v) will be attached to the proper $c \in V_C \cup \{Thing\}$ vertex and will be labelled as $\mathcal{L}_{AUG}(c, v) = \mathcal{L}(c) \in L_A$, while the newly inserted vertex $v \in V_U$ will not be labelled.
- If $m_i \in L_R$ or $m_i \in L_C$, do nothing as they are already part of the schema-guide graph G_{SCM}.

4.2 Query Pattern Graph Formulation

Definition 5. *The Query Pattern Graph G_{QP} is the minimal connected subgraph of the augmented schema-guide graph G_{AUG} such that it includes all the matched elements $m_i \in c$ and there exists no other query pattern graph G'_{QP} such that $C(G'_{QP}) < C(G_{QP})$ for a given cost function C.*

In our framework, we define the cost function as the average pairwise distance between the matched elements m_i, in other words $C(G_{QP}) = \sum \mathrm{dist}(m_i, m_j)$ $\forall i, j = 1, \ldots, n$ and $i \neq j$. In order to minimize the cost function for every pair of matched elements (m_i, m_j) $i \neq j$, we calculate their shortest path in G_{AUG}. Note that during the shortest path calculations we ignore the directionality of the edges. Moreover, since a matched element m_i, i.e. a source or sink of the shortest path algorithm, can also be an edge, then the distance between two matched elements counts the number of both vertices and edges that needs to traverse across the augmented summary graph G_{AUG}. Finally, we combine all the pairwise shortest path forming a connected subgraph, the Graph Query Pattern G_{QP}. Note that the G_{QP} is a "compressed" form of graph pattern G_Q. Each node $v \in \{V_C \cup \{Thing\}\}$ from G_{QP} is a hypernode that corresponds to a single subtree $G'_Q \subset G_Q$ which contains two vertices $v_1, v2 \in G'_Q$, where $v_1 \in V_{VAR}$ and $v_2 \in V_C$ connected with an edge (v_1, v_2) with label *type*. Any edge e initially attached to hypernode v can be attached to vertex v_1.

4.3 Query Mapping

The next step is to translate the query pattern graph G_{QP} into a SPARQL query. Remember that $G_{QP} = (V_{QP}, E_{QP}, \mathcal{L}_{QP})$ is a subgraph of the augmented schema-guide graph G_{AUG} and it is also a compressed form of a graph pattern G_Q, thus some of its elements have known fixed values and some need to be associated with variables. More specifically, vertices $v \in V_C \cup V_U$ represent generic RDF elements and need to be associated with variables. Note also that the labels of the vertices can be used as constants in the triple patterns, while the labels of the edges as predicates. Given these observations, to produce conjunctive SPARQL queries for every graph element $\in G_{QP}$ we perform:

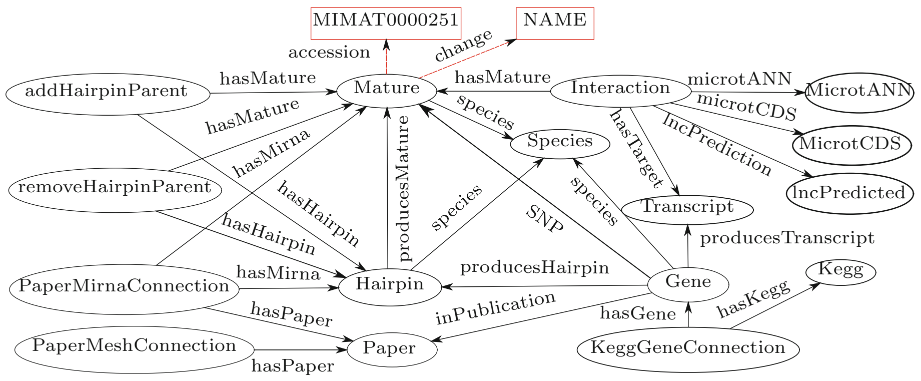

Fig. 3. One of the 6 possible Augmented schema-guide graph for the keywords *MIMAT0000251, name* and *hasTarget* in the DIANA dataset.

- if $v \in \{V_U \cup V_{VAL}\}$, then associate the vertex v with a new variable $\texttt{var}(v)$,
- if $v \in V_C$, then associate the vertex v with a new variable $\texttt{var}(v)$ and produce the triple pattern $\texttt{var}(v)$ `rdf:type` $\mathcal{L}_{QP}(v)$, where $\mathcal{L}_{QP}(v) \in L_C$.
- if $(s, o) \in E$ from vertex $s \in V_C$ to vertex $o \in V_C$ represents an inter-entities property where $\mathcal{L}_{QP}((s, o)) \in L_R$, then produce the triple pattern $\texttt{var}(s)$ $\mathcal{L}_{QP}(e)$ $\texttt{var}(o)$,
- if $(s, o) \in E$ from vertex $s \in V_C$ to vertex $o \in V_{VAL}$ represents an entity-to-attribute property where $\mathcal{L}_{QP}(o) \in L_V$ and $\mathcal{L}_{QP}((s, o)) \in L_A$, then produce the triple pattern $\texttt{var}(s)$ $\mathcal{L}_{QP}((s, o))$ $\texttt{var}(o)$. `FILTER(`$\texttt{var}(o) = \mathcal{L}_{QP}(o)$`)` and
- if $(s, o) \in E$ from vertex $s \in V_C$ to vertex $o \in V_U$ represents an entity-to-attribute property, such that $\mathcal{L}_{QP}((s, o)) \in L_A$, then produce the triple pattern $\texttt{var}(s)$ $\mathcal{L}_{QP}((s, o))$ $\texttt{var}(o)$.

Finally, all produced queries are ranked based on a given ranking function $\mathcal{R}$. In our framework, we provide three different ranking functions (a) the number of triplet patterns, (b) the average shortest path distance and (c) the longest shortest path distance. The former works on the SPARQL form of the generated query, while the latter two work on the query pattern graph G_{QP}. Note that the smaller the score of the ranking function $\mathcal{R}$ for a given query, the higher the ranking position. The idea behind this is based on the assumption known as "Locality of Information", meaning that the information required by the user can be modelled in terms of entities which are closely related [11].

4.4 Example

Let's assume that the user is interested in exploring the DIANA dataset and has provided the following as keywords: *MIMAT0000251, name, hasTarget.* Given the subgraph depicted in Fig. 1, the keywords match to the following elements: (a) *MIMAT0000251* to the literal `MIMAT0000251` that is connected via the property `accession` met only with RDF entities of `Mature` type, (b) *name* to the literal `NAME` that is connected via the property `change` met with RDF entities of type either `Mature` and to the entity-to-attribute property `name` met with entities of type `Hairpin`, `Mature`, `Species` and `Gene`, resulting in 5 possible matches, (c) *hasTarget* to the inter-entities property `hasTarget` met with subject of type `Interaction` and object of type `Transcript`.

Overall we have $1 \times 5 \times 1 = 5$ possible combinations of the keywords to matched elements, that would result to 5 possible queries. Let's consider one combination, where the *name* keyword matches to the literal "NAME". Figure 3 shows the augmented schema-guide graph G_{AUG} for this combination. From this G_{AUG} we calculate the shortest paths between all pairs of matched elements and since we have 3 keywords, we need to calculate of $\binom{3}{2} = 3$ shortest paths. We then combine them into a single connected component, generating the Query Pattern graph G_{QP} shown in Fig. 4. Note that the extra node `Transcript` is attached to the property `hasTarget` in order to form a complete triple pattern, although it is not part of neither of the previous calculated shortest paths. Finally, the G_{QP} is mapped to the SPARQL query shown in Fig. 5.

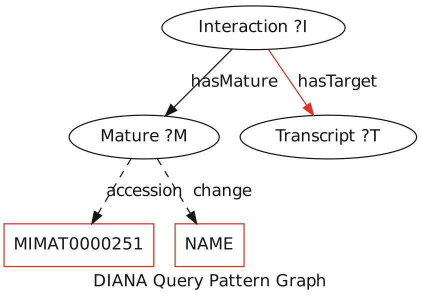

Fig. 4. The Query Pattern Graph extrapolated from the Augmented Schema-Guide Graph of Fig. 3. In red we depict the matched elements (Color figure online).

```
SELECT ?I ?M ?T WHERE
{
    ?I a diana:Interaction.
    ?M a diana:Mature.
    ?T a diana:Transcript.
    ?I diana:hasMature ?M.
    ?I diana:hasTarget ?T.
    ?M diana:accession ?a.
    FILTER( ?a = "MIMAT0000251").
    ?M diana:change ?n.
    FILTER(?n = "NAME").
}
```

Fig. 5. The generated SPARQL query from the Query Pattern Graph of Fig. 4.

5 Temporal Operators

When working with diachronic data, querying should also involve temporal constraints. In our method, we support the following three temporal operators: (a) `at` (b) `before` and (c) `after`. The first one can be used to retrieve data at a specific time point, while the other two can be used to define a time window constraint. The temporal operators are used as follows: `property operator:value`, where `property` is the temporal entity-to-attribute property that the selected

Table 2. Aggregated statitics for the datasets of AI4B and DIANA.

Dataset	# Triples	# Classes	# Properties	# Unique String Values
AI4B	$2,7 \times 10^6$	15	148	6.350
DIANA	$4,6 \times 10^9$	16	76	613.408

temporal operator will be applied and `value` is a desired value. For example, if the user provides as input "*paper let7a year after:2006*", she wants to retrieve publications related with miRNAs named "let7a" published after 2006.

When a temporal operator is used in a query, we consider the selected temporal property as an extra keyword and proceed to the construction of the query pattern graph G_{QP} and its mapping to SPARQL as described in Sect. 4. There are only two small differences. The first one is that when we construct the augmented schema-guide graph G_{AUG}, we add, apart from the temporal property itself, an extra node representing the desired value provided by the user. The second difference is that when we translate the temporal entity-to-attribute property (recall that it will be represented as an edge (s, o) within G_{QP} from vertex $s \in V_C$ to vertex o that represents the selected value) we produce the following triplet $\texttt{var}(s)$ $\mathcal{L}_{QP}(e)$ $\texttt{var}(o)$. followed by (a) $\texttt{FILTER(var}(o) = \mathcal{L}_{QP}(o))$, when the `at` temporal operator is used, (b) $\texttt{FILTER(var}(o) \leq \mathcal{L}_{QP}(o))$ when the `before` temporal operator is used and (c) $\texttt{FILTER(var}(o) \geq \mathcal{L}_{QP}(o))$, when the `after` temporal operator is used.

6 Evaluation

We evaluated our proposed keyword search method using two RDF datasets, the AI4B[3] and the DIANA[4]. The AI4B dataset contains information about biomass products, while the diachronic DIANA dataset contains aggregated information of the miRNA world from well-known biology databases that change and evolve throughout their lifespan. Table 2 shows detailed information about the characteristics of the two datasets. The implementation of the presented method for the DIANA dataset is available at http://snf-624527.vm.okeanos.grnet.gr:8080/KeywordSearchDiana/. Finally, our approach has also been incorporated in the collaborative platform LinkZoo [7].

To evaluate our approach we perform an effectiveness study. We have asked our collaborators to provide keyword queries along with a natural language description of the required information. We have aggregated 20 queries in total, 15 for the DIANA (Q1-Q15) and 5 for the AI4B (Q16-Q20). An example query is *"'Alzheimer's disease' mature version at:18"* and the corresponding description is *"Retrieve all mature miRNAs of miRBase version 18 that are related with Alzheirmer's disease"*. All queries used in the evaluation can be found at https://web.imis.athena-innovation.gr/redmine/projects/lodgov/wiki/Deliverable2_4_Evaluation.

[3] http://snf-629975.vm.okeanos.grnet.gr:8897/ai4b/sparql.
[4] http://leonardo.imis.athena-innovation.gr:8891/diana/sparql.

The effectiveness is calculated by the *Reciprocal Rank* metric defined as $RR = 1/r$, where r is the ranking position of the query that corresponds to the provided natural language description. To further assist our collaborators in the evaluation process, we provide also a natural language description of the generated SPARQL queries by incorporating the verbalization system SPARQL2NL [8]. Figure 6 shows the Reciprocal Rank we have calculated for our three ranking functions. In the 17 out of 20 queries, we got an RR of 1 meaning that we were able to get the information requested by the users.

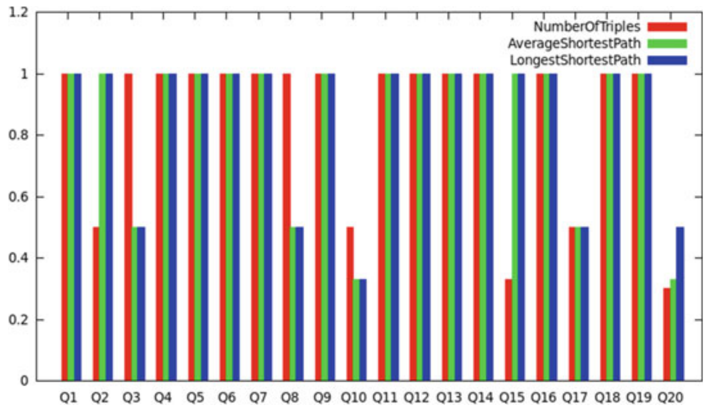

Fig. 6. Reciprocal Rank of different ranking functions on DIANA and AI4B datasets.

7 Related Work

The keyword search problem over structured data, either tree structured [4,6] or graph structured [1,2,5], is a problem that has widely been explored. These works involve the following basic steps: (a) mapping the keyword elements to structured data elements (b) connect the keyword elments by searching for substructures on the data, and (c) return as output the retrieved substructures given a scoring function. Contrary to the previous approaches, Tran et al. [10] proposed a different solution for the keyword search problem. Instead of computing the answers directly on the data, they compute structured queries allowing the user to choose the appropriate one. The advantages of this process are the valuable information provided by the queries allows better comprehension of the retrieved results and the exploitation of the existing query optimization techniques. Our approach keywords-to-sparql queries follows [10] approach, but enriches the information stored within indices and also uses a different exploratory method. More specifically, in comparison with Tran et al.'s keyword index our term index maintains also information about RDF classes and inter-entities properties allowing to efficiently track possible matches of keywords under a uniform space, while our augmented schema-guide graph allows also the encoding of temporal properties. Furthermore, we create multiple augmented schema-guide graphs one per keywords combination and use the notion of shortest paths to create a query pattern graph.

8 Conclusion

In this paper, we have presented a novel method for keyword search on data modeled under the RDF graph representation. Our approach can also be applied to generic graph-structured data if a schema can be extracted. Contrary to the most common approaches found in bibliography for the keyword search problem, where answers are directly computed from the data, our algorithm generates structured queries driven by the schema of the data. This leverages a two-fold advantage. Firstly, it provides valuable information to the user in terms of comprehension of the data. Secondly, it profits from the system optimization in order to extrapolate the required results. Furthermore, our method is enriched with temporal operators allowing a more efficient and deeper exploration of the diachronic data within predefined time points. We have evaluated our approach under an effectiveness study using two real datasets and we have achieved an excellent performance capturing the information requested by the users.

Acknowledgements. This study has been supported by LODGOV project, Research Programme ARISTEIA (EXCELLENCE), General Secretariat for Research and Technology, Ministry of Education, Greece and the European Regional Development Fund.

References

1. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword searching and browsing in databases using BANKS. In: ICDE, pp. 431–440 (2002)
2. Bikakis, N., Giannopoulos, G., Liagouris, J., Skoutas, D., Dalamagas, T., Sellis, T.: RDivF: diversifying keyword search on RDF graphs. In: Aalberg, T., Papatheodorou, C., Dobreva, M., Tsakonas, G., Farrugia, C.J. (eds.) TPDL 2013. LNCS, vol. 8092, pp. 413–416. Springer, Heidelberg (2013)
3. Brickley, D., Guha, R.V.: RDF Schema 1.1 W3C Recommendation, 25 February 2014. www.w3.org/TR/rdf-schema/
4. Cohen, S., Mamou, J., Kanza, Y., Sagiv, Y.: XSEarch: a semantic search engine for XML. In: VLDB, pp. 45–56 (2003)
5. He, H., Wang, H., Yang, J., Yu, P.S.: BLINKS: ranked keyword searches on graphs. In: SIGMOD, pp. 305–316 (2007)
6. Kimelfeld, B., Sagiv, Y.: Finding and approximating Top-k answers in keyword proximity search. In: PODS, pp. 173–182. ACM (2006)
7. Meimaris, M., Alexiou, G., Gkirtzou, K., Papastefanatos, G., Dalamagas, T.: RDF resource search and exploration with LinkZoo. In: DATA. p. (2015) (to appear)
8. Ngonga Ngomo, A.C., Bühmann, L., Unger, C., Lehmann, J., Gerber, D.: Sorry, I Don'T Speak SPARQL: Translating SPARQL Queries into Natural Language. In: WWW, pp. 977–988 (2013)

9. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C Recommendation (2008). http://www.w3.org/TR/rdf-sparql-query/
10. Tran, D.T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. In: ICDE (2009)
11. Tran, T., Cimiano, P., Rudolph, S., Studer, R.: Ontology-based interpretation of keywords for semantic search. In: Aberer, K., et al. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 523–536. Springer, Heidelberg (2007)