Exploiting Semantic Result Clustering to Support Keyword Search on Linked Data

Ananya Dass[1], Cem Aksoy[1], Aggeliki Dimitriou[2], and Dimitri Theodoratos[1]

[1] New Jersey Institute of Technology, USA
[2] National Technical University of Athens, Greece

Abstract. Keyword search is by far the most popular technique for searching linked data on the web. The simplicity of keyword search on data graphs comes with at least two drawbacks: difficulty in identifying results relevant to the user intent among an overwhelming number of candidates and performance scalability problems. In this paper, we claim that result ranking and top-k processing which adapt schema unaware IR-based techniques to loosely structured data are not sufficient to address these drawbacks and efficiently produce answers of high quality. We present an alternative solution which hierarchically clusters the results based on a semantic interpretation of the keyword instances and takes advantage of relevance feedback from the user. Our clustering hierarchy exploits graph patterns which are structured queries clustering together result graphs of the same structure and represent possible interpretations for the keyword query. We present an algorithm which computes r-radius Steiner patterns graphs using exclusively the structural summary of the data graph. The user selects relevant pattern graphs by exploring only a small portion of the hierarchy supported by a ranking of the hierarchy components.Our experimental results show the feasibility of our system by demonstrating short reach times and efficient computation of the relevant results.

1 Introduction

In recent years, the proliferation of semistructured data (e.g., XML and RDF data) has sparked a lot of interest on developing techniques for effectively and efficiently querying tree and graph data. Keyword search is, by far, the most popular technique for querying data with loose structure on the web. Its success comes from the flexibility it provides to the user to retrieve information from a data source without mastering a complex query language (e.g., XQuery, SPARQL) and without knowing the structure of the data source. The advantages of keyword search on semistructured data come with a number of disadvantages. Keyword queries are ambiguous in determining both: the user intent and the form of the results. For this reason, keyword search on tree and graph data faces three major challenges:
(a) **Determining the form of the results:** In contrast to the IR domain where the answer of a keyword query is a set of flat documents, in the domain of tree and graph databases, each result in the keyword query answer

is a database substructure (e.g., node, subtree, subgraph). This not only multiplies enormously the number of candidate results and, therefore, makes the evaluation more complex, but it also raises the issue of determining an appropriate form for the results. The goal is to return results (substructures) which are meaningful to the user. Different approaches on tree data define the results as LCA nodes [24, 4], minimum connecting trees [12], instance trees [2], etc. In the context of graph data multiple approaches return trees [23, 8, 13] usually constraint by the adopted search algorithms. Indeed, traditional keyword search algorithms on graphs associate keywords only to vertices and, by construction, compute and return minimum spanning trees [3, 14, 11, 9]. However, tree structures do not appropriately capture the semantics of queries on graph data which should naturally return graph structures. Further, in RDF data, semantics are assigned to graph elements through their association to schema elements. This information should be taken into account in the search process and integrated in the query results in order to help disambiguating the queries and their results. In this direction, some approaches attempt to exploit predicates [22, 7, 13].
(b) **Identifying the relevant results.** Because of the ambiguity of keyword queries there is usually an overwhelming number of results matching the query keywords (candidate results) of which only a tiny portion is relevant to the user intent. Multiple approaches assign semantics to keyword queries by exploiting structural and semantic features of the data in order to automatically filter out irrelevant results [20]. Although filtering approaches are intuitively reasonable, they are sufficiently ad-hoc and they are frequently violated in practice resulting in low precision and/or recall. A better technique followed by some other approaches ranks the candidate results in descending order of their estimated relevance [10, 20]. Given that users are typically interested in a small number of query results, some of these approaches combine ranking with top-k algorithms for keyword search [23, 22, 8]. Keyword search over graph data returns a multitude of candidate results and of extended diversity. Therefore, current algorithms compute candidate results in an approximate way by considering only those which maintain the keyword instances in close proximity [3, 14, 5, 11, 9, 18, 21, 15]. The ranking and top-k processing of the filtered results usually employ IR-style metrics for flat documents (e.g., tf*idf or PageRank) [10, 23, 22, 7, 20] adapted to the structural characteristics of the data. However, the occurrence statistics alone cannot capture effectively the diversity of the results represented in a large graph dataset neither identify the intent of the user. As a consequence the produced rankings are, in general, of low quality.
(c) **Coping with the performance scalability issue**. As mentioned above, the number of candidate results can be very large. Computing all the results of a certain form is intractable. For instance, the problem of finding the Steiner trees for a set of keywords in a data graph is NP-complete [15]. The current algorithms which compute all the results of a certain form restricted so that their size is below a certain threshold are still of high complexity. Consequently, these algorithms do not scale satisfactorily when the size of the data graph and the number of query keywords increases.

Our approach. In this paper, we present a novel approach for keyword search on RDF graph data. Our approach utilizes a semantic two-level hierarchical clustering of the keyword query results. The first—fine-grained—level of clustering, partitions the results based on pattern graphs. These pattern graphs, when used as queries on the RDF graph, compute the results of the corresponding cluster. The second—coarser—level of clustering, partitions the patterns (and their results) based on the different types of construct (e.g., class, property, value) each query keyword matches. However, our approach does not exhaustively generate and cluster all the results and the hierarchy components. Instead, it benefits from relevance feedback at different levels of granularity to identify the pattern graphs which are relevant to the user intent. The hierarchy components are presented to the user ranked to facilitate their selection. The candidate pattern graphs are generated using the structural summary of the data graph (a concept analogous to that of 1-index in tree databases [16]) without accessing the data graph. Only the selected pattern graphs are evaluated on the data returning all and only the relevant results. This way, our system addresses efficiently the problems of relevant result identification and performance scalability.

Contribution. The main contributions of the paper are the following:

- We define query results as meaningful subgraphs of the data graph that appropriately connect together keyword *matching constructs* (elementary subgraphs representing semantic interpretations of the keyword instances). This addresses the problem (a) of determining the form of query result (Section 2.)
- We use the concept of patterns graph of a keyword query to cluster together result graphs that share the same structure. The pattern graphs involve all the matching constructs of the query keywords on the structural summary and express the possible interpretations of the keyword query on the graph data (Section 3.1).
- We design an algorithm which takes as input the matching constructs of the query keywords on the structural summary and computes patterns forming r-radius Steiner graphs that involve these matching constructs. Our algorithm computes the patterns on the structural summary without accessing the data (Section 3.2).
- We design a clustering hierarchy for the results which is based on pattern graphs and keyword matching constructs. Our clustering hierarchy allows the user to disambiguate the query and compute the relevant results while examining only a small portion of the hierarchy components. To shorten the user interaction we devise ranking techniques for the hierarchy components that take into account structural and semantic information and occurrence frequency statistics (Section 4).
- We implemented and experimentally evaluated our approach. Our results on measuring reach time show that the user can find the relevant patterns in short time supported effectively by our ranking of the hierarchy components, and that the system efficiently computes the required hierarchy components on the structural summary and evaluates the relevant pattern graph on the data (Section 5).

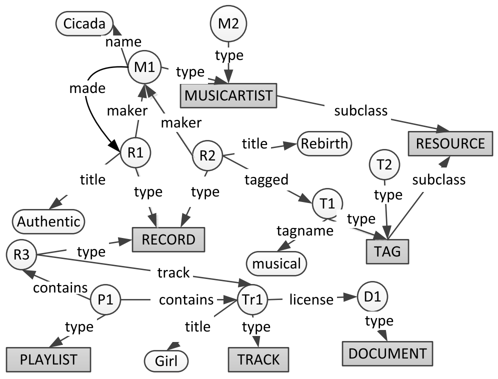


Fig. 1: An RDF graph

2 Data Model and Query Language Semantics

2.1 Data Model

Resource Description Framework (RDF) provides a framework for representing information about web resources in a graph form. The RDF vocabulary includes elements, that can be broadly classified into Classes, Properties, Entities and Relationships. All the elements are resources. RDF has a special class, called *Resource* class and all the resources that are defined in an RDF graph belong to the *Resource* class. Our data model is an RDF graph defined as follows:

Definition 1 (RDF Graph). An *RDF graph* is a quadruple $G = (V, E, L, l)$ where:

V is a finite set of vertices, which is the union of three disjoint sets: V_E (representing entities), V_C (representing classes) and V_V (representing values).

E is a finite set of directed edges, which is the union of four disjoint sets: E_R (inter-entity edges called *relationship* edges which represent entity relationships), E_P (entity to value edges called *property* edges which represent property assignments), E_T (entity to class edges called *type* edges which represent entity to class membership) and E_S (class to class edges called *subclass* edges which represent class-subclass relationship).

L is a finite set of labels that includes the labels "type", "subclass" and "resource".

l is a function from $V_C \cup V_V \cup E_R \cup E_P$ to L. That is, l assigns labels to class and values vertices and to relationship and property edges.

Every entity belongs to a class. Figure 1 shows an example RDF graph (a subgraph of the Jamendo Dataset[3]).

[3] http://dbtune.org/jamendo/

2.2 Queries and Answers

A *query* is a set of keywords. The *answer* of a query Q on an RDF graph G is a set of subgraphs (*result graphs*) of G, where each result graph involves at least one instance of every keyword in Q. A *keyword instance* of a keyword k in Q is a vertex or edge label containing k. In order to facilitate the interpretation of the semantics of the keyword instances, every instance of keyword in a query is matched against a small subgraph (*matching construct*) of the graph G which involves this keyword instance. Each matching construct provides a deeper insight about the context of a keyword instance in terms of classes, entities and relationship edges. We link one matching construct for every keyword in the query Q through edges (*inter-construct connections*) and common vertices into a connected component to form a *result graph.*

Definition 2 (Matching Construct). Given a keyword k of a query and an RDF graph G, for every instance of k in G, we define a *matching construct* as a small subgraph of G. If the instance i of k in G is:

- the label of a class vertex $v_c \in V_C$, the matching construct of i is the vertex v_c (*class matching construct*).
- the label of a value vertex $v_v \in V_V$, the matching construct of i comprises the value vertex v_v, the corresponding entity vertex, and its class vertices along with the property and type edges between them (*value matching construct*).
- the label of relationship edge $e_r \in E_R$, the matching construct of i comprises the relationship edge e_r, its entity vertices and their class vertices along with the type edges between them (*relationship matching construct*).
- the label of property edge $e_p \in E_P$, the matching construct of i comprises the property edge e_p, its value and the entity vertices, and the class vertices of the entity vertex along with the type edges between them (*property matching construct*).

Figures 2(a), (b), (c) and (d) show a class, value, relationship and property matching construct, respectively, for different keyword instances in the RDF graph of Figure 1. Underlined labels in a matching construct denote the keyword instances on which the matching construct is defined (called *active keyword instances* of the matching construct).

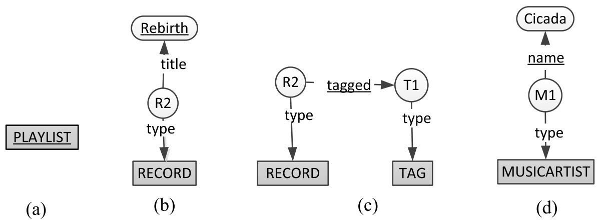


Fig. 2: Matching Constructs for (a) class, (b) value, (c) relationship and (d) property.

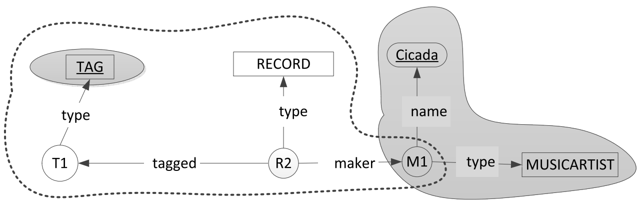


Fig. 3: Inter-construct connection

Definition 3 (Query Signature). Given a query Q and an RDF graph G, a *signature* of Q is a function from the keywords of Q that matches every keyword k to a matching construct of k in G.

Figures 2(a), (b), (c) and (d) show a query signature for the query {*Playlist, Rebirth, tagged, name*}. Note that a signature of a query Q can have less matching constructs than the keywords in Q, since one matching construct can have more than one active keyword instance.

Definition 4 (Inter-construct Connection). Given a query signature S, an *inter-construct connection* between two distinct matching constructs C_1 and C_2 in S is a simple path augmented with the class vertices of the intermediate entity vertices in the path (if not already in the path) such that: (a) one of the terminal vertices in the path belongs to C_1 and the other belongs to C_2, and (b) no vertex in the connection except the terminal vertices belong to a construct in S.

Figure 3 shows an inter-construct connection between the matching constructs for keywords *Tag* and *Cicada* in the RDF graph of Figure 1. The matching constructs are shaded and the inter-construct connection is circumscribed with a dotted line.

In order to define result graphs, we need the concept of acyclic subgraph with respect to a query signature. Let G_s be a subgraph of the RDF graph that comprises all the constructs in the signature of a query. We construct an undirected graph G_c as follows: there is exactly one vertex in G_C for every matching construct and for every vertex not in a matching construct in G_s. Further: (a) if v_1 and v_2 are non-construct vertices in G_c, there is an edge between v_1 and v_2 in G_c iff there is an edge between the corresponding vertices in G_s, (b) If v_1 is a construct vertex and v_2 is a non-construct vertex in G_c, there is an edge between v_1 and v_2 in G_c iff there is an edge between a vertex of the construct corresponding to v_1 and the vertex corresponding to v_2 in G_s, and (c) if v_1 and v_2 are two construct vertices, there is an edge between them in G_c iff there exists in G_s an edge between a vertex of the construct corresponding to v_1 and a vertex of the construct corresponding to v_2 and that edge does not occur in any one of the constructs. Graph G_s is said to be *connection acyclic* if there is no cycle in G_c.

Consider the query $Q = \{$*Cicada, musical, Playlist*$\}$ on the RDF graph G of Figure 1. Figure 4 shows two subgraphs of G which comprise a signature of

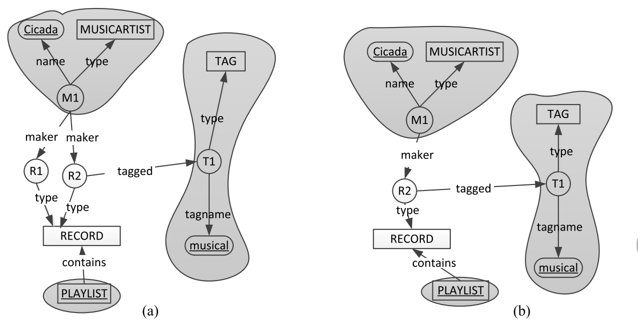


Fig. 4: (a) Invalid result graph (b) Valid result graphs

Q on G. The active keyword instances are underlined and the corresponding matching constructs are shaded. One can see that the subgraph in Figure 4(a) is connection cyclic while the other subgraph Figure 4(b) is connection acyclic.

Definition 5 (Result Graph). Given a signature S for a query Q over an RDF graph G, a *result graph* of Q for S is a connected connection acyclic subgraph G_R of G which contains only the matching constructs in S and possibly inter-construct connections between them.

Therefore, a result graph of a query contains all the matching constructs of a signature of the query and guarantees that they are linked with inter-construct connections into a connected whole. Note that a result graph might not contain any inter-construct connection (this can happen if every matching construct in the query signature overlaps with some other matching construct). However, if inter-construct connections are used within the result graph, no redundant (cycle creating) inter-construct connections are introduced.

We now define the *answer* of a query Q on an RDF graph G as the set of result graphs of Q on G.

3 Computing Pattern Graphs on the Structural Summary

We formally introduce in this section the structural summary of a data graph and query pattern graphs. Then, we present an algorithm for computing pattern graphs on a structural summary.

3.1 The Structural Summary and Pattern Graphs

In order to construct pattern graphs we use the structural summary of the RDF graph. Intuitively, the structural summary of an RDF graph G is a special type of graph which summarizes the data graph showing vertices and edges corresponding to the class vertices and property, relationship and subclass edges in G.

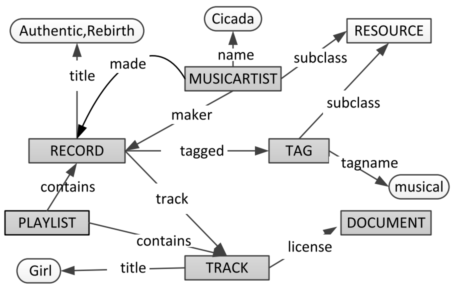


Fig. 5: Structural Summary

Definition 6 (Structural Summary). The *structural summary* of an RDF graph $G(V, E, L, l)$ is a vertex and edge labeled graph $G'(V', E', L', l')$ where

- $V' = V'_C \cup V'_v$ where:
  (a) V'_C is a set of class vertices which has a one to one mapping f onto V_C,
  (b) V'_v is a set of value vertices which contains a vertex for every distinct pair (c, l_p) such that there exists an entity of class c in the RDF graph G having a property labeled by l_p.
  A class vertex c in V_C is labeled by the label of the corresponding class vertex $f(c)$ in G.
- $E' = E'_p \cup E'_r \cup E'_s$ where:
  (a) E'_p is a set of edges from vertices in V'_c to vertices in V'_v such that there is an edge $(c, v) \in E'_p$ labeled by l_p iff there is an entity of class $f(c)$ in G which has a property edge labeled by l_p,
  (b) E'_r is a set of edges from a vertex in V'_C to another vertex in V'_C such that there is an edge $(c_1, c_2) \in E'_r$ labeled by l_r iff there is an edge from an entity of $f(c_1)$ to an entity of $f(c_2)$ in G labeled by l_r.
  (c) E'_s is a set of edges from a vertex in V'_C to another vertex in V'_C such that there is an edge $(c_1, c_2) \in E'_s$ labeled by *subclass* iff there is an edge from $f(c_1)$ to $f(c_2)$ in G labeled by *subclass*.
- L' is the set of labels of vertices in V' and edges in E'.
- l' is a function assigning labels to vertices and edges in G' as already described above.

Figure 5 shows the structural summary for the RDF graph G of Figure 1. Similarly to matching constructs on the data graph, we define matching constructs on the structural summary and we refer to them as MC. The pattern graphs comprise MCs for every keyword in the query and the connections between them.

The next objective is to compute subgraphs of the structural summary, strictly consisting of one matching construct for every keyword in the query Q and the connections between them without forming any cycle. These subgraphs are called *result pattern graphs.*

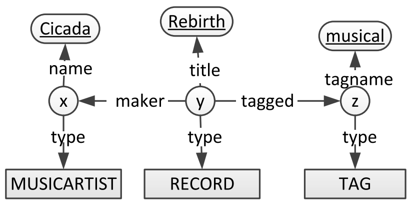


Fig. 6: Query Pattern Graph

Definition 7 (Pattern Graph). A *(result) pattern graph* for a keyword query Q is a graph similar to a result graph for Q, and with the same keyword instances, but with the following two exceptions

(a) the labels of the entity vertices in the result graph, if any, are replaced by distinct variables in the pattern graph. These variables are called *entity variables* and they range over entity labels.

(b) The labels of the value vertices are replaced by distinct variables whenever these labels are not the keyword instances in the result graph. These variables are called *value variables* and they range over value labels in the RDF graph.

Figure 6 shows an example of a pattern graph, for the keyword query $Q = \{$*Cicada, Rebirth, musical*$\}$. X, Y and Z are entity variables.

3.2 An Algorithm for Computing Query Pattern Graphs

Our algorithm computes r-radius Steiner pattern graphs. The r-radius Steiner graph computation is inspired by the algorithm of [18]. However, unlike that algorithm, our algorithm allows the keywords to match the edges labels of the graph.

Our algorithm is shown in algorithm 1. It takes as input the structural summary G' of a data graph and a signature S of a query Q on G' (the set of MCs for the keywords of Q on G'). It produces as output the r-radius Steiner query pattern graphs on G' that contain S whose radius r is minimum.

Every MC in S is associated with one or two class vertices. Set $\mathcal{K}$ is initialized with all the class vertices in S. The adjacency matrix A is a square matrix of order $n+1$, where n is the number of class vertices in G'. The first row and column of A record the class identifiers. $\mathcal{A}(i,j)$ holds information about the relationship edges connecting the classes $\mathcal{A}(i,0)$ and $\mathcal{A}(0,j)$. Initially the algorithm tries to find Steiner pattern graphs with radius $r = 1$. If no graph is found, r is gradually increased. Sets $\mathcal{P}$ and $\mathcal{C}$ represent the data structures recording the pattern graphs and their corresponding connecting vertices, respectively. Procedure *GeneratePattern* generates the pattern graphs using the information recorded about the relationship edges between the connecting vertex and the class vertices in $\mathcal{K}$ and the MCs in S.

Algorithm 1 Query Pattern Graphs Computation

```
Input: $S$: signature, $G'$: structural summary.
Output: $\mathcal{P}$: set of pattern graphs.
 1: $\mathcal{K} \leftarrow$ extract distinct class vertices from $S$;
 2: $\mathcal{A} \leftarrow$ adjacency matrix encoding class vertices $V'_C$ and rel. edges $E'_R$ in $G'$;
 3: $r \leftarrow 1$;                                   ▷ radius of the Steiner graph
 4: $\mathcal{P} \leftarrow \emptyset$;
 5: $\mathcal{C} \leftarrow \emptyset$;                ▷ set of connecting vertices
 6: while $\mathcal{C} = \emptyset$ do
 7:     for all rows $i \in \mathcal{A}$ do
 8:         $conn \leftarrow \mathcal{A}(i, 0)$;        ▷ adding class id to set conn
 9:         for all cols $j \in \mathcal{A}$ do
10:             if $\mathcal{A}(i, j) \neq \emptyset$ then
11:                 $conn = conn \cup (\mathcal{A}(0, j))$;    ▷ adding class id to set conn
12:         if $\mathcal{K} \subseteq conn$ then
13:             $\mathcal{C} \leftarrow \mathcal{A}(i, 0)$;    ▷ a connecting node is found
14:             $P \leftarrow$ GeneratePattern$(G', r, \mathcal{S}, \mathcal{A}(i, 0))$;
15:             if $P \notin \mathcal{P}$ then
16:                 $\mathcal{P} \leftarrow \mathcal{P} \cup P$;
17:         $conn \leftarrow \emptyset$;
18:     if $\mathcal{C} = \emptyset$ then
19:         r=r+1;
20:         $\mathcal{A}' \leftarrow$ modify adjacency matrix $\mathcal{A}$ to represent how the class vertices are
    connected to each other at a distance less than or equal to r;
21:         $\mathcal{A} \leftarrow \mathcal{A}'$
```

4 Semantic Hierarchical Clustering and Ranking

We now describe our result clustering hierarchy, how the user navigates through the hierarchy and graph ranking.

Semantic hierarchical clustering. The hierarchy has two levels on top of the result graph layer. The pattern graphs of a query Q on an RDF graph G define a partition of the result graphs of Q on G. The pattern graphs constitute the *first level* of the clustering hierarchy. Multiple pattern graphs can share the same signature. The signatures determine a partition of the pattern graphs of Q on G. They, in turn, define a partition of the results which is coarser than that of the pattern graphs. The signatures constitute the *second (top) level* of the hierarchy.

Hierarchy navigation. In order to navigate through the hierarchy after issuing a query the user starts from the top level. The top level may have numerous signatures. However, the user does not have to examine all the signatures. Instead, she is presented with the MC list for one of the query keywords. We describe below how this list of MCs is ranked. As mentioned earlier, the MCs of a keyword provide all the possible interpretations for this keyword in the data. The user selects the MC that she considers relevant to her intent. Subsequently, she is presented with the MC lists of the other query keywords, though some of the MC lists can be skipped. This can happen if the user selects an MC which involves more than one keyword instances that she wants to see combined together in one MC. Once MCs for all keywords have been selected, that is, a query signature has been determined, the system presents a ranked list of all the pattern graphs that comply with the signature. The user chooses the pattern graph of her preference which is evaluated by the system. The result graphs are returned to the user.

Ranking. The MCs for a keyword are ranked in an MC list based on the following rules: (a) MCs that involve more than one active keyword instances are ranked first in order of the number of active keyword instances they contain, (b) class MCs, relationship MCs and property MCs are ranked next in that order, (c) value MCs follow next and are ranked in descending order of the frequency of their value. The *value frequency* f_m^v of a value MC m with property p, class c and value (keyword) v is the number $n_{p,c}^v$ of occurrences of the value v in matching constructs involving p and c in the data graph divided by the number $n_{p,c}$ of occurrences of property matching constructs in the data graph involving p and c. That is,

$$f_m^v = n_{p,c}^v / n_{p,c}$$

This ranking of the MCs favors MCs with multiple keyword instances based on the assumption that keywords that occur in close proximity are more relevant to the user's intent. Further, it favors MCs whose active keyword matches a schema element (class, relationship or property), favoring most class MCs which have unique occurrences in the data graph. Finally, value MCs are ranked at the end since they are more specific. The value frequency of a value MC reflects the popularity of this MC in the data. Therefore, value MCs with high value frequency are ranked higher than value MCs for the same value with low value frequency.

The pattern graphs the system ranks share the same signature. Thus, they are r-radius graphs with the same r. In almost all the cases they have the same number of edges and they differ only in the relationship edges which are not part of any MC. For this reason, the pattern graphs are ranked in descending order of their connecting edge frequency defined next. Given a pattern P, its *connecting edge frequency*, $f_c(P)$, is the sum of the number n_e occurrences in the data graph of the relationship edges e in P that do not occur in an MC in P divided by the total number $|E_R|$ of relationship edges in the data graph. That is, if E_c is the set of these relationship edges in P,

$$f_c(P) = \sum_{e \in E_c} n_e / |E_R|$$

In order to rank MCs and pattern graphs our system needs statistics about value MCs and their property edges and about connecting relationship edges in pattern graphs. This information is precomputed and stored with the structural summary when this one is constructed. Therefore, no access to the data graph is needed.

Query	keywords	#I	#MC	#S
1	Track Obsession Divergence format title mp32	699	29	2,646
2	biography guitarist track lemonade	633	18	216
3	Knees Cicada recorded_as	59	7	9
4	sweet recorded_as Signal onTimeLine 104734	177	17	48
5	Track Nuts chillout ACExpress	618	21	252
6	Mako La deux date love time	2846	36	24,300
7	Fantasie recorded_as factor published_as format date title	145	25	588
8	Fantasie recorded_as Performance Paure	68	8	8
9	Briareus Vampires Infirmary Cool	154	18	288
10	Fantasie text Paure Document	144	13	42

Table 1: Queries used in the experiments and their statistics

5 Experimental Evaluation

We implemented our approach and run experiments to evaluate our system. The goal of our experiment is to assess: (a) the effectiveness of our clustering approach, (b) the efficiency of our techniques in providing suggestions to the user and in obtaining results from the selected pattern graphs in real time. It is not meaningful to run experiments to measure precision and recall since our approach exploits relevance feedback and returns all and only the results which are relevant to the user intent (perfect precision and recall).

Dataset and queries. We use Jamendo, a large repository of Creative Commons licensed music. Jamendo is a dataset of 1.1M triples and of 85MB size containing information about musicians, music tracks, records, licenses of the tracks, music categories, track lyrics and many other details related to them. ts structural summary was extracted and stored in a relational database. Experiments are conducted on a standalone machine with an Intel i5-3210M @ 2.5GHz processors and 8GB memory.

Users provided different queries on the Jamendo dataset and navigated through our hierarchical clustering system to select a relevant MC (for every keyword) and a relevant pattern graph (when more than one were proposed by the system for the selected MCs). We report on 10 of them. The queries cover a broad range of cases. They involve from 3 to 7 keywords. Table 1 shows the keyword queries and statistics about them. For every query it shows the total number of keyword instances in the data graph (#I), the number of MCs (for all the keywords) in the structural summary (#MC), and the total number of signatures (#S). As we can see, a query can have many pattern graphs (their number is greater than or equal to that of the signatures). However, thanks to our hierarchical clustering approach, the user has to examine only one or at most two of them. Further, exploiting our ranking of the MCs, the user has to examine only a fraction of the MCs for every query.

Effectiveness of hierarchical clustering. In order to measure the retrieval effectiveness of our hierarchical clustering, we adapted the *reach time* metric used in [17, 19]. The reach time sets forth to quantify the time spent by a user to locate the relevant results. In our case, the relevant results are represented by the relevant pattern graph. The relevant results in terms of subgraphs of the data graph can then be retrieved by evaluating the pattern graph against the database. For simplicity, we assume that the user always selects one relevant pattern graph. We employ different versions of reach time in two different settings: when the components (MCs and pattern graphs) are ranked, and when they are not. rt_{avg} and rt_{max} apply to the case the components are not ranked. rt_{avg} (resp. rt_{max}) denotes the average (resp. maximum) number of components the user needs to examine in order to retrieve the relevant pattern. rt_{rank} denotes the number of components the user examines in order to retrieve the relevant pattern when the components are ranked. For instance, if a query has k keywords and the user needs to examine m_i of the ranked MCs for keyword i and p of the ranked pattern graphs for the selected MCs,

$$rt_{rank} = \sum_{i=1}^{k} m_i + p$$

Figure 7 shows the reach times for the queries of Table 1.

As one can see, the user has to examine on the average a small number of components even for queries with many keywords. Further, when the components are ranked, rt_{rank} is smaller than rt_{avg} in most cases and never comparable to rt_{max}. This demonstrates the feasibility of our hierarchical clustering system and the quality of the component ranking process.

Efficiency of the system. In order to asses the efficiency of out system, we measured the time needed to compute the ranked list of MCs for the query keywords and the time needed to evaluate the selected pattern graphs.

Figure 8(a) shows the total time (*totalMC*) needed to compute and rank the MCs for the keyword queries. It also shows the shortest(*minMC*) and the longest

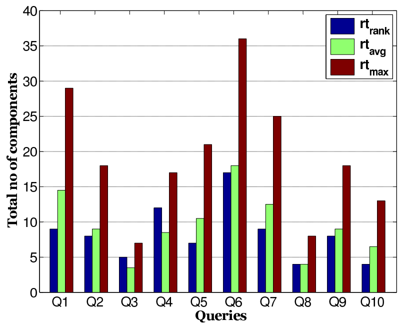
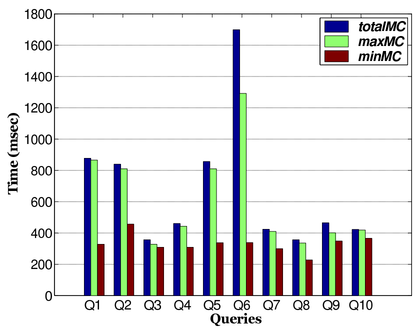


Fig. 7: Reach times of the two clustering approaches



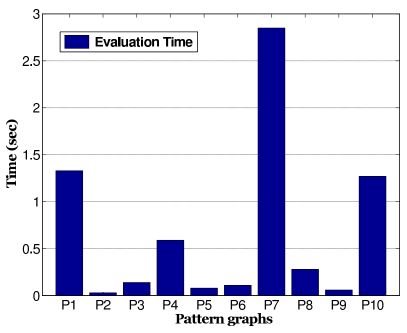


Fig. 8: (a) Time to compute the MCs for the query keywords (b) Evaluation time for the selected pattern graphs

(*maxMC*) time needed to compute the rank list of MCs of a keyword in a given query. The list of MCs for the first keyword in the query is presented to the user for selection as soon as it is computed. The plot displays interactive time for the all the queries which do not delay the selection process. The time needed to compute the pattern graphs on the structural summary after the MCs for the query keywords are selected by the user are insignificant and are not displayed here. This is expected since the pattern graphs are computed using exclusively the structural summary whose size is very small compared to the size of the data.

Figure 8(b) displays the time needed to evaluate the selected pattern graphs on the data graph. This diagram again shows interactive times even though it is a prototype system and no optimizations have been applied.

6 Related Work

Several algorithms explore data graphs to support keyword search. In [3] a backward search algorithm BANK is presented for finding Steiner trees. The Steiner tree problem is NP complete. Hence different techniques are used to work around NP completeness. In [5], a dynamic programming approach applicable to only few keywords and having an exponential time complexity is employed. In [9] a polynomial delay algorithm is introduced. The algorithm in [14] produced trees rooted at distinct vertices which was supplemented by BLINK [11] with an efficient indexing structure. Tree-based methods produce succinct answers but answers from graph-based methods are more informative. Recent graph based approaches like [18] compute all possible r-radius Steiner graphs and index them. This method is prone to produce redundant results since it is possible that a highly ranked r-radius Steiner graph is included in another Steiner graph having a larger radius. [21] finds multi-centered subgraphs called communities containing all the keywords, such that there exists at least one path of distance less than or equal to R_{max} between every keyword instance and a center vertex. Later in

[15], r-cliques containing all the keywords are found such that the distance between any two pair of keyword matching vertices is no more than r. Finding r-clique with the minimum weight is an NP-hard problem, hence the authors provided an algorithm with polynomial delay to find the top-k r-cliques where r is an input to the algorithm. Predicting an optimal r for producing r-cliques is a challenge because it is possible that there exists no clique with that r or less.

Unlike traditional graph-based keyword search, on RDF graphs, keywords can match both a vertex and an edge label of the graph [22, 6]. Some approaches exploit a summary of the underlying RDF graph to find query patterns [23, 22, 8]. Other approaches directly search for connections between keyword matches on the data graph [6] for a given keyword query. None of these approaches exploits hierarchical clustering and relevance feedback. User interaction for predicate selection is explored in [13] but without leveraging pattern graphs and structural summaries. Hierarchical clustering mechanisms and user interaction at multiple levels are discussed in [19, 1]. However, these studies focus on tree-structured XML data and do not consider graphs.

7 Conclusion

We have presented a novel approach to address the problems related to keyword search on large RDF data. Our approach hierarchically clusters the result graphs and leverages relevance feedback from the user. In order to form the clustering hierarchy, we use matching constructs and pattern graphs which are subgraphs representing semantic interpretations for keywords and queries, respectively. We presented an algorithm to efficiently compute r-radius Steiner pattern graphs. All hierarchy components are computed efficiently on the structural summary without accessing the (much larger) data graph. We presented a technique that ranks the hierarchy components based on their structural and semantic features and occurrence frequencies. Our approach allows the user to explore only a tiny portion of the clustering hierarchy in selecting the relevant graph patterns supported by the component ranking. The experimental evaluation of our approach shows its feasibility by demonstrating short reach times to the relevant graph patterns and efficient computation of the relevant result graphs on the data graph.

As a future work we are planning to study relaxation techniques for the selected relevant pattern graphs in order to enable the extraction of loosely connected result graphs which are still of interest to the user.

References

1. C. Aksoy, A. Dass, D. Theodoratos, and X. Wu. Clustering query results to support keyword search on tree data. In *WAIM*, pages 1–12, 2014.
2. C. Aksoy, A. Dimitriou, D. Theodoratos, and X. Wu. Xreason: A semantic approach that reasons with patterns to answer XML keyword queries. In *DASFAA*, pages 299–314, 2013.

3. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.
4. A. Dimitriou and D. Theodoratos. Efficient keyword search on large tree structured datasets. In *KEYS*, pages 63–74, 2012.
5. B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, pages 836–845, 2007.
6. S. Elbassuoni and R. Blanco. Keyword search over RDF graphs. In *CIKM*, pages 237–242, 2011.
7. S. Elbassuoni, M. Ramanath, R. Schenkel, and G. Weikum. Searching RDF graphs with sparql and keywords. *IEEE Data Eng. Bull.*, pages 16–24, 2010.
8. H. Fu, S. Gao, and K. Anyanwu. Disambiguating keyword queries on RDF databases using "Deep" segmentation. In *ICSC*, pages 236–243, 2010.
9. K. Golenberg, B. Kimelfeld, and Y. Sagiv. Keyword proximity search in complex data graphs. In *SIGMOD*, pages 927–940, 2008.
10. L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. Xrank: ranked keyword search over XML documents. In *SIGMOD*, pages 16–27, 2003.
11. H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.
12. V. Hristidis, N. Koudas, Y. Papakonstantinou, and D. Srivastava. Keyword proximity search in XML trees. *IEEE Trans. Knowl. Data Eng.*, pages 525–539, 2006.
13. M. Jiang, Y. Chen, J. Chen, and X. Du. Interactive predicate suggestion for keyword search on RDF graphs. In *ADMA (2)*, pages 96–109, 2011.
14. V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.
15. M. Kargar and A. An. Keyword search in graphs: Finding r-cliques. *VLDB*, pages 681–692, 2011.
16. R. Kaushik, P. Bohannon, J. F. Naughton, and H. F. Korth. Covering indexes for branching path queries. In *SIGMOD Conference*, pages 133–144, 2002.
17. K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *WWW*, pages 658–665, 2004.
18. G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008.
19. X. Liu, C. Wan, and L. Chen. Returning clustered results for keyword search on XML documents. *IEEE Trans. Knowl. Data Eng.*, pages 1811–1825, 2011.
20. Z. Liu and Y. Chen. Processing keyword search on XML: a survey. *World Wide Web*, pages 671–707, 2011.
21. L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE*, pages 724–735, 2009.
22. T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (RDF) data. *ICDE*, pages 405–416, 2009.
23. H. Wang, K. Zhang, Q. Liu, T. Tran, and Y. Yu. Q2semantic: A lightweight keyword interface to semantic search. In *ESWC*, pages 584–598, 2008.
24. Y. Xu and Y. Papakonstantinou. Efficient LCA based keyword search in XML data. In *EDBT*, pages 535–546, 2008.