COLD. Revisiting Hub Labels on the database for large-scale graphs

Alexandros Efentakis[1], Christodoulos Efstathiades[1,2] and Dieter Pfoser[3]

[1] Research Center “Athena”
`efentakis@imis.athena-innovation.gr`
[2] Knowledge and Database Systems Laboratory
National Technical University of Athens, Greece
`cefstathiades@dblab.ece.ntua.gr`
[3] Department of Geography and GeoInformation Science, George Mason University
`dpfoser@gmu.edu`

Abstract. Shortest-path computation is a well-studied problem in algorithmic theory. An aspect that has only recently attracted attention is the use of databases in combination with graph algorithms to compute distance queries on large graphs. To this end, we propose a novel, efficient, pure-SQL framework for answering exact distance queries on large-scale graphs, implemented entirely on an open-source database system. Our COLD framework (COmpressed Labels on the Database) may answer multiple distance queries (vertex-to-vertex, one-to-many, kNN, RkNN) not handled by previous methods, rendering it a complete solution for a variety of practical applications in large-scale graphs. Experimental results will show that COLD outperforms previous approaches (including popular graph databases) in terms of query time and efficiency, while requiring significantly less storage space than previous methods.

1 Introduction

Answering distance queries on graphs is one of the most well-studied problems on algorithmic theory, mainly due to its wide range of applications. Although a lot of recent research focused exclusively on transportation networks (cf. [9] for the most recent overview) the emergence of social networks has generated massive unweighted graphs of interconnected entities. On such networks, the distance between two vertices is an indication of the closeness of their entities, i.e., for finding users closely related to each other or extracting information about existing communities within the social media users. Although we may always use a breadth first search (BFS) to calculate the distance between any two vertices on such graphs, that approach cannot facilitate fast-enough queries on main memory or be easily adapted to secondary storage solutions.

Moreover, most of the excellent preprocessing techniques available for road networks cannot be adapted to large-scale graphs, such as social or collaboration networks. So far, the most promising approach for this type of graphs builds on the 2-hop labeling or hub labeling (HL) algorithm [23],[12], in which we store a two-part label $L(v)$ for every vertex v: a forward label $L_f(v)$ and a backward label $L_b(v)$. These labels are then used to very fast answer vertex-to-vertex shortest-path queries. This technique has been adapted successfully to road networks [2, 3, 15, 4] and quite recently has also been extended to undirected, unweighted graphs [5, 14, 25]. The HL method has also been applied for one-to-many, many-to-many and kNN queries in road networks [16, 17] and kNN and RkNN queries in the context of social networks in [21].

Although hub labeling is an extremely efficient shortest-path computation method using main memory, there are very few works that try to replicate those algorithms for secondary storage. HLDB [18] stores the calculated hub labels for continental road networks in a commercial database system and translates the typical HL distance query between two vertices to plain SQL commands. Moreover, it showed how to efficiently answer kNN queries and k-best via points, again by means of SQL queries. Recently, HopDB [25] proposed a customized solution that utilizes secondary storage also during preprocessing. Unfortunately, both methods have their shortcomings. HLDB has only been tested on road networks and consequently small labels sizes (<100). Its speed would seriously degrade for large-scale graphs due to the much larger label size. HopDB answers only vertex-to-vertex queries and is a customized C++ solution that cannot be used with existing database systems and, hence, has limited practical applicability.

This work presents a database framework that may service multiple distance queries on massive large-scale graphs. Our pure-SQL *COLD* framework (COmpressed Labels on the Database) can answer multiple exact distance queries (point-to-point, kNN) in addition to RkNN and *one-to-many* queries not handled by previous methods, rendering it a complete database solution for a variety of practical massive, large-scale graph problems. Our extensive experimentation will show that COLD outperforms previous solutions, including specialized graph databases, on all aspects (including query performance and memory requirements), while servicing a larger variety of distance queries. In addition, COLD is implemented using a popular, open-source database engine with no third-party extensions and, thus, our results are easily reproducible by anyone.

The outline of the remainder of this work is as follows. Section 2 presents related work. Section 3 describes the novel COLD framework and its implementation details. Experiments establishing the benefits of COLD are provided in Section 4. Finally, Section 5 gives conclusions and directions for future work.

2 Related work

Throughout this work we use undirected, unweighted graphs $G(V, E)$ (where V represents vertices and E arcs). A k-Nearest Neighbor (kNN) query seeks the k-nearest neighbors to an input vertex q. The RkNN query (also referred as the monochromatic RkNN query), given a query point q and a set of objects P, retrieves all the objects that have q as one of their k-nearest neighbors according to a given distance function $dist()$. In graph networks, $dist(s, t)$ corresponds to the minimum network distance between the two objects. Formally RkNN(q) =

$\{p \in P : dist(p,q) \leq dist(p,p_k)\}$ where p_k is the k-Nearest Neighbor (kNN) of p. Throughout this work, we assume that objects are located on vertices and we always refer to *snapshot* kNN and RkNN queries on graphs, i.e, objects are not moving. Also, similarly to previous works, the term *object density* D refers to the ratio $|P|/|V|$, where P is a set of objects in the graph and $|V|$ is the total number of vertices. Although, there is extensive literature focusing on kNN and RkNN queries in Euclidean space, since our work focuses on graphs we will only describe related work focusing on the latter.

Regarding road networks and kNN queries, G-tree [33] is a balanced tree structure, constructed by recursively partitioning the road network into sub-networks. Unfortunately, this method cannot scale for continental road networks, since it requires several hours for its preprocessing. Moreover, it requires a *target selection phase* to index which tree-nodes contain objects (requiring few seconds) and thus, cannot be used for moving objects. Recently, the work of [17] expanded the graph-separators CRP algorithm of [13] to handle kNN queries on road networks. Unfortunately, (i) CRP also requires a target selection phase and thus, cannot be applied to moving objects and (ii) it may only perform well for objects near the query location. Hence, this solution is also not optimal. The latest work for kNN queries on road networks is the SALT framework [22] which may be used to answer multiple distance queries on road networks, including *vertex-to-vertex* (v2v), single source (one-to-all, range, one-to-many) and kNN queries. This work expands the graph-separators GRASP algorithms of [20] and the ALT-SIMD adaptation [19] of the ALT algorithm and offers very fast preprocessing time and excellent query times. For kNN queries, SALT does not require a target selection phase and hence it may be used for either static or moving objects.

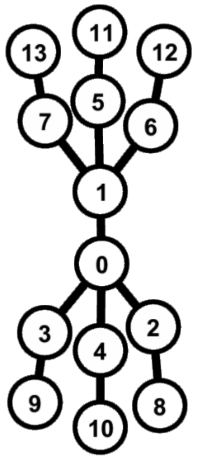
For RkNN queries on road networks, the work of [30] uses Network Voronoi cells (i.e., the set of vertices and arcs that are closer to the generator object) to answer RkNN queries. This work has only been tested on a relatively small network ($110K$ arcs) and all precomputed information is stored in a database. Despite the fact that the preprocessing stage for computing the Network Voronoi cells is quite costly, the queries' executions times range from $1.5s$ for $D = 0.05$ and $k = 1$, up to $32s$ for $k = 20$, rendering this solution impractical for real-time scenarios. Up until recently, the only work dealing with other graph classes (besides road networks) is [32], although it has only been tested on sparse networks, e.g., road networks, grid networks (max degree 10), p2p graphs (avg degree 4) and a very small, sparse co-authorship graph ($4K$ nodes). In this work, the conducted experiments for values of $k > 1$ refer only to road networks, therefore the scalability of this work for denser graphs and larger values of k is questionable. Recently, Borutta et al. [10] extended this work for time-dependent road networks, but presented results were not very encouraging. The larger road network tested had $50k$ nodes (queries require more than $1s$ for $k = 1$) and for a network of $10k$ nodes and $k = 8$, RkNN queries take more than $0.3s$ (without even adding the I/O cost). In a nutshell, all existing contributions and methods have not been tested on dense, large-scale graphs, cannot scale for increasing k values and their performance highly depends on the object density D.

Our work builds upon the 2-hop labeling or Hub Labeling (HL) algorithm of [23, 12] in which, preprocessing stores at every vertex v a forward $L_f(v)$ and a backward label $L_b(v)$. The forward label $L_f(v)$ is a sequence of pairs $(u, dist(v,u))$, with $u \in V$. Likewise, the backward label $L_b(v)$ contains pairs $(w, dist(w,v))$. Vertices u and w are denoted as the *hubs of v*. The generated labels conform to *the cover property*, i.e., for any s and t, the set $L_f(s) \cap L_b(t)$ must contain at least one hub that is on the shortest $s-t$ path. For undirected graphs $L_b(v) = L_f(v)$. To find the network distance $dist(s,t)$ between two vertices s and t, a HL query must find the hub $v \in L_f(s) \cap L_b(t)$ that minimizes the sum $dist(s,v) + dist(v,t)$. By sorting the pairs in each label by hub, this takes linear time by employing a coordinated sweep over both labels. The HL technique has been successfully adapted for road networks in [2, 3, 15, 4]. In the case of large-scale graphs, the Pruned Landmark Labeling (PLL) algorithm of [5] *produces a minimal labeling for a specified vertex ordering.* In this work, vertices are ordered by degree, whereas the work of [14] improves the suggested vertex ordering and the storage of the hub labels for maximum compression. The HL method has also been used for one-to-many, many-to-many and kNN queries on road networks in [16] and [17] respectively. Our latest work [21] proposed *ReHub*, a novel main-memory algorithm that extends the Hub Labeling approach to efficiently handle RkNN queries. The main advantage of the *ReHub* algorithm is the separation between its costlier offline phase, which runs only once for a specific set of objects and a very fast online phase which depends on the query vertex q. Still, even the costlier offline phase hardly needs more than $1s$, whereas the online phase requires usually less than $1ms$, making *ReHub* the only RkNN algorithm fast enough for real-time applications and big, large-scale graphs.

Regarding secondary-storage solutions, Jiang et al. [25] propose their HopDB algorithm that suggest an efficient HL index construction when the given graphs and the corresponding index are too big to fit into main memory. The work of [1] introduced the HLDB system, which answers distance and kNN queries in road networks entirely within a database by storing the hub labels in database tables and translating the corresponding HL queries to SQL commands. Throughout this work, we will compare our proposed COLD framework to HLDB, since to the best of our knowledge, it is the only framework that may answer exact distance queries entirely within a database. Moreover, within the COLD framework we also adapt our *ReHub* main-memory algorithm into a database context, so that its online phase may be translated to fast and optimized SQL queries.

3 Contribution

This section presents the *COLD* (COmpressed Labels on the Database) database framework. COLD can answer multiple distance queries (vertex-to-vertex, kNN, RkNN and *one-to-many*) for large-scale graphs using SQL commands. Since COLD builds on HLDB [1] and *ReHub* [21], we will follow the notation and running example presented there, for highlighting the necessary concepts and challenges for adapting those previous works, (i) in the context of large-scale



Vertex	Hub Labels (h,d)
0	(0,0)
1	(0,1), (1,0)
2	(0,1), (2,0)
3	(0,1), (3,0)
4	**(0,1), (4,0)**
5	(0,2), (1,1), (5,0)
6	(0,2), (1,1), (6,0)
7	(0,2), (1,1), (7,0)
8	(0,2), (2,1), (8,0)
9	(0,2), (3,1), (9,0)
10	**(0,2), (4,1), (10,0)**
11	(0,3), (1,2), (5,1), (11,0)
12	**(0,3), (1,2), (6,1), (12,0)**
13	(0,3), (1,2), (7,1), (13,0)

Fig. 1 & Table 1: A sample Graph G and the created hub-labels

graphs for [1] and (ii) within the boundaries of a relational database management system (RDBMS) for [21]. To this end, we chose PostgreSQL [29] for our implementation, given that it is a popular, open-source RDBMS. Although we use some PostgreSQL-specific data-types and SQL extensions, we do not use any third-party extensions but only features included in its standard installation.

3.1 Implementation

The COLD framework assumes that we have a correct hub labeling (HL) framework that generates hub-labels for the undirected, unweighted graphs we wish to query. Although COLD will work with any correct HL algorithm, in this work we use the [6] implementation of the PLL algorithm of [5] to generate the necessary labels. To highlight the results of this process, the labels for the undirected, unweighted graph G of Figure 1 are shown in Table 1. Throughout this work, we will refer to those labels as the *forward labels*. The forward label $L(v)$ for a vertex v is an array of pairs $(u, dist(v, u))$ sorted by hub u. Since our work also focuses on snapshot kNN and RkNN queries, there also some objects $P \in V$ that do not change over time. For our specific running example we assume that $P = \{4, 10, 12\}$ and thus, we highlight the respective entries of Table 1.

Vertex-to-Vertex (v2v) queries. To find the network distance $dist(s, t)$ between two vertices s and t, a HL query must find the hub $v \in L(s) \cap L(t)$ that minimizes the sum $dist(s, v) + dist(v, t)$. For our sample graph G, the minimum distance between e.g., vertices 2 and 7 is $d(2, 7) = 3$, using the hub 0. To translate this HL query into SQL commands, in HLDB [1] forward labels are stored in a database table denoted *forward* where the labels of vertex v are stored as triples of the form $(v, hub, dist(v, hub))$ (see Table 2). The table *forward* has the

Table 2: The *forward* table used in HLDB for the sample graph G

v	hub	dist
...	...	...
2	0	1
2	2	0
...	...	...
7	0	2
7	1	1
7	7	0
...	...	...

Table 3: The *forwcold* table used for COLD for the sample graph G

v	hubs	dists
...	...	...
2	$\{0,2\}$	$\{1,0\}$
...	...	...
7	$\{0,1,7\}$	$\{2,1,0\}$
...	...	...

Code 1.1: V2v query for HLDB

```
SELECT MIN(n1.dist+n2.dist)
FROM forward n1, forward n2
WHERE n1.v = s
AND n2.v = t
AND n1.hub = n2.hub;
```

Code 1.2: V2v query for COLD

```
SELECT MIN(n1.d+n2.d) FROM
/* Expand hubs, dists arrays */
(SELECT UNNEST(hubs) AS hub,
UNNEST(dists) AS d
FROM forwcold WHERE v = s) n1,
(SELECT UNNEST(hubs) AS hub,
UNNEST(dists) AS d
FROM forwcold WHERE v = t) n2
WHERE n1.hub=n2.hub;
```

combination of (v, hub) as the primary key and is clustered according to those columns, so that "*all rows corresponding to the same label are stored together to minimize random accesses to the database*" [1]. Then we can find the distances between any two vertices s and t by the SQL query of Code 1.1.

Although the HLDB vertex-to-vertex (v2v) query is very simple, there is one major drawback. For such a query, HLDB has to fetch from secondary storage the subset of $|L(s)| + |L(t)|$ rows with common hubs. Although this is practical for road networks where the forward labels have less than 100 hubs per vertex [3], it cannot scale for large-scale graphs where the forward labels have thousand of hubs per vertex. Moreover, on such graphs the *forward* DB table and the corresponding primary key index will become too large, which is also an important disadvantage. To this end, we take advantage of the fact that PostgreSQL features an array data type that allows columns of a DB table to be defined as variable-length arrays. Hence, in COLD we store hubs and distances for a vertex (both ordered by hub) as arrays in two separate columns (i.e., hubs and dists) in a single row. The resulting *forwcold* compressed DB table is shown in Table 3. This approach not only emulates exactly how labels are stored on main-memory for fast v2v queries but also has considerable advantages: (i) The *forwcold* DB table has exactly $|V|$ rows (ii) The *forwcold* DB table has the column v as primary key without needing a composite key. This alone facilitates faster queries. Moreover the size of the corresponding index will be much smaller. In fact, our experimentation will show that the primary-key index for *forwcold* may be $> 4,400\times$ smaller than the index size of HLDB. (iii) For a v2v

Table 4: Necessary data structures for the sample graph G, $P = \{4, 10, 12\}$ and *one-to-many*, kNN and RkNN queries

Hub	**Backward Labels (to-many) [16]**	**kNN Backward Labels (k=2) [1]**	**RkNN Backward Labels (k=1) [21]**	**Obj.**	**kNN Result (k=1) (Obj., dist) [21]**
0	(4,1), (10,2), (12,3)	(4,1), (10,2)	(4,1), (12,3)	**4**	(10,**1**)
1	(12,2)	(12,2)	(12,2)		
4	(4,0), (10,1)	(4,0),(10,1)	(4,0), (10,1)	**10**	(4,**1**)
6	(12,1)	(12,1)	(12,1)		
10	(10,0)	(10,0)	(10,0)	**12**	(4,**4**)
12	(12,0)	(12,0)	(12,0)		

query, COLD needs to access exactly two rows, regardless of the sizes of $|L(s)|$ and $|L(t)|$. This way, we efficiently minimized the secondary-storage utilization, even working inside a database. The resulting SQL query for COLD is shown in Code 1.2. There we exploit the fact that PostgreSQL "*guarantees that parallel unnesting*" for hubs and distances for each nested query "*will be in sync*", i.e., each pair (hub, dist) is expanded correctly since for the same v the respective arrays have the same number of elements[4].

Additional queries overview. For answering more complex (kNN, RkNN and *one-to-many*) distance queries on a HL framework for a set of objects P, we need to build some additional data structures from the forward labels (for undirected graphs). Then to answer the respective query we only need to combine the forward labels $L(q)$ of query vertex q, with the respective data structure explained in the following. Those data structures are summarized in Table 4.

For answering *one-to-many* queries, i.e., calculate distances between a source vertex q and all objects in P, we need to build the *backward labels-to-many* by basically ordering the forward labels of the objects by hub [16] and then by distance for the same hub. For kNN queries we only need to keep at most the k-best pairs (of smallest distances) per hub from the backward labels-to-many to create the *kNN backward labels* [1]. In our specific example, the kNN backward labels for $k = 2$ and hub 0, do not contain the pair $(12, 3)$. Finally, for RkNN queries, we must first calculate the *kNN Results* (i.e., the NN of the object 4 is the object 10 with distance 1) and then we build the RkNN backward labels, based on the observation that *"we need to access those pairs from the backward labels-to-many to a specific object, if and only if those distances are equal or smaller than the distance of the kNN of this object"* [21]. In our specific example, the RkNN backward labels for $k = 1$ and hub 0, do not contain the pair (10,2) since the NN of object 10 (the object 4) is within distance 1. Although for our small graph the differences between the individual data structures seem minimal, for larger graphs those differences become very prominent. This was also showcased by the theoretical analysis provided in [21] which showed that backward labels-to-many will have on average $D \cdot |HL|$ pairs, the kNN backward labels have at most $k \cdot |V|$ pairs and the RkNN backward labels have on average $\varepsilon \cdot D \cdot |HL|$

[4] http://stackoverflow.com/a/23838131

Table 5: The *knntab* table used in HLDB for the sample graph G, $k = 2$ and $P = \{4, 10, 12\}$

hub	dist	obj
0	1	4
0	2	10
1	2	12
...	...	...

Table 6: The *knntab* table used in COLD for the sample graph G, $k = 2$ and $P = \{4, 10, 12\}$

hub	dist	objs
0	1	{4}
0	2	{10}
1	2	{12}
...	...	...

pairs where ε may be < 0.01 for specific datasets and experimental settings. Moreover, Efentakis et al. [21] have shown how these additional data structures may be constructed from the forward labels in main-memory, requiring less than few seconds, even for the larger tested datasets.

***k*NN queries.** To translate the HL kNN query into SQL, HLDB stores *kNN backward labels* in a separate DB table denoted *knntab* that stores triples of the form $(hub, dist, obj)$ (see Table 5). The respective table *knntab* has the combination of $(hub, dist, obj)$ as a composite primary key and is clustered according to those columns. Note that in HLDB, we cannot use the combination of $(hub, dist)$ as a primary key, because especially in large scale graphs we will have a lot of distance ties even for k-entries for the same hub. Then we can can answer a kNN query from vertex q by the SQL query of Code 1.3. Again, the kNN HLDB query has the same drawbacks as before, i.e., it has to retrieve $|L(q)|$ rows from *forward* and $k \cdot |L(q)|$ rows from *knntab* tables, for a total of $(k+1) \cdot |L(q)|$ rows retrieved from secondary storage. Moreover in a database, it makes sense to create one large *knntab* table for the maximum value *kmax* of k (e.g., for $k = 16$) that may be serviced by the DB framework and that same table will be used for all kNN queries up to $k = kmax$. In that case, the HLDB framework will have to retrieve $(kmax + 1) \cdot |L(q)|$ rows for every kNN query regardless of the value of k.

To remedy the HLDB drawbacks, COLD creates the *knncold* DB table (Table 6) that has the columns $(hub, dist, objs)$, whereas objects are grouped and ordered per hub and distance (the column *objs* is an array). Although for our sample graph G, the DB tables *knntab* and *knncold* seem identical, COLD's method offers several advantages: (i) We can now use the combination of $(hub, dist)$ as a primary key, which makes the respective index significantly smaller and faster and (ii) In case of many distance ties (common to large-scale graphs) and one large *knncold* DB table that services all kNN queries for values of k up to the maximum value $kmax$, we only need to fetch the first *k-objs* entries (i.e., `objs[1:k]`) per hub and dist, which makes the later sorting faster (see Code 1.4).

One-to-many queries. Similar to how COLD handles kNN queries, for one-to-many queries, COLD stores the *backward labels-to-many* in a new *objcold* DB table that has an identical format to *knncold*, i.e., it has three columns $(hub, dist, objs)$ whereas objects are grouped and ordered per hub and distance. *Objcold* also uses the combination of $(hub, dist)$ as a primary key. The resulting

Code 1.3: kNN query for HLDB

```
SELECT MIN(n1.dist+n2.dist),
n2.obj FROM
forward n1, knntab n2
WHERE n1.v = q
AND n1.hub = n2.hub
GROUP BY n2.obj
ORDER BY MIN(n1.dist+n2.dist)
LIMIT k;
```

Code 1.4: kNN query for COLD

```
SELECT MIN(n1.d+n2.dist),
UNNEST(objs) AS obj FROM
(SELECT UNNEST(hubs) AS hub,
UNNEST(dists) AS d
FROM forwcold WHERE v = q) n1,
/* k-entries per hub,dist */
(SELECT hub, dist,objs[1:k]
FROM knncold) n2
WHERE n1.hub=n2.hub
GROUP BY obj
ORDER BY MIN(n1.d+n2.dist)
LIMIT k;
```

one-to-many query (Code 1.5) is quite similar to COLD's kNN query, but (i) it operates on the larger *objcold* DB table (ii) It does not have the `ORDER BY ... LIMIT k` clause and (iii) We use the entire *objs* array per hub and distance instead of `objs[1:k]`. Note that HLDB cannot possibly support such queries because it will need to retrieve on average $|L(q)|$ rows from the *forward* table and a total of $|L(q)| \cdot D \cdot (|HL|/|V|)$ [21] rows from the corresponding *objlab* table, which will be prohibitively slow for very large datasets.

Table 7: The *knnres* table used in COLD for RkNN queries, the sample graph G, $k = 1$ and $P = \{4, 10, 12\}$

obj	dists	objs
4	{1}	{10}
10	{1}	{4}
12	{4}	{4}

R*k*NN queries. For RkNN queries, COLD stores the RkNN backward labels in a separate *revcold* DB table that has an identical format to previous *knncold* and *objcold* DB tables, i.e., three columns $(hub, dist, objs)$ where objects are grouped and ordered per hub and distance and the combination of $(hub, dist)$ used as a primary key. COLD also stores the *kNN Results*, i.e., the kNN of all objects in another *knnres* DB table that has the format $(obj, dists, objs,)$ where obj is the primary key and *objs* and *dists* are arrays (both ordered by distance). Therefore the kNN of object p is the `objs[k]` within distance `dists[k]` of the respective row for p. Again it makes sense to build a *knnres* DB table for a max value of *kmax* that may service RkNN queries for varying values of k. As a result, during the RkNN COLD query, we will have to use an additional `JOIN` between the *revcold* and *knnres* DB tables. The resulting query is shown in Code 1.6.

We see that even the more complex RkNN query in COLD requires just a few lines of SQL code that will work on any recent PostgreSQL version without any need of third-party extensions or specialized index structures. In fact, all DB tables in COLD, use only standard B-tree primary key indexes, without any modifications. To satisfy this strict requirement, we effectively compressed

Code 1.5: *One-to-many* COLD query

```
SELECT MIN(n1.d+n2.dist),
UNNEST(objs) AS obj FROM
(SELECT UNNEST(hubs) AS hub,
UNNEST(dists) AS d
FROM forwcold
WHERE v = q) n1,
objcold n2
WHERE n1.hub=n2.hub
GROUP BY obj;
```

Code 1.6: RkNN query for COLD

```
SELECT n3.id2,n3.dist FROM
/* n3 subquery is a modified
 one-many-query to revcold */
(SELECT MIN(n1.d+n2.dist) AS d3,
UNNEST(objs) AS obj FROM
(SELECT UNNEST(hubs) AS hub,
UNNEST(dists) AS d
FROM forwcold WHERE v = q) n1,
revcold n2
WHERE n1.hub=n2.hub
GROUP BY obj
ORDER BY obj,MIN(n1.d+n2.dist)
) n3,
/* Join with knnres table */
(SELECT obj, dists[k] AS dist
FROM knnres) n4
WHERE n3.obj=n4.obj
AND n3.d3<=n4.dist
ORDER BY n3.obj;
```

the index sizes by grouping rows per vertex (*forcold* table) or object (*knnres* table), or by hub and distance for *knncold*, *objcold* and *rknncold*. And although we used PostgreSQL specific SQL extensions for expanding the stored arrays, latest versions of other databases (e.g., Oracle) support similar array data-types. Hence, it would be quite easy to port COLD to other database vendors as well.

This section detailed the COLD framework in terms of design and implementation. COLD can answer multiple distance queries (v2v, kNN, RkNN and one-to-many) based on data stored in an off-the-shelf relational database. We also presented the actual queries used and the way the necessary data structures are stored within the database, so that our results are easily reproducible. Although we focused on query efficiency, it is important to note that once we create the *forcold* table, all the adjoining DB tables within COLD may also be created using SQL commands (resulting queries were omitted due to space restrictions). This fact also shows that COLD is truly a pure-SQL framework for servicing multiple distance queries on large-scale graphs. We also provided the necessary theoretical details as to why the COLD framework will outperform existing solutions. This will be further quantified in the following section.

4 Experimental Evaluation

To assess the performance of COLD on various large-scale graphs, we conducted experiments on a workstation with a 4-core Intel i7-4771 processor clocked at 3.5GHz and 32Gb of RAM, running Ubuntu 14.04. We compare our COLD

Table 8: Networks graphs statistics

Graph	$\mid V \mid$	$\mid E \mid$	Avg degr.	$\mid HL \mid / \mid V \mid$	PLL Preproc. Time (s)
Facebook	4,039	88,234	22	26	0.03
NotreDame	325,729	1,090,108	3	55	6
Gowalla	196,591	950,327	5	100	13
Youtube	1,134,890	2,987,624	3	167	123
Slashdot1	77,360	469,180	6	204	11
Slashdot2	82,168	504,230	6	216	13
Citeseer1	268,495	1,156,647	4	408	110
Amazon	334,863	925,872	3	689	230
DBLP	540,486	15,245,729	28	3,628	5,720
Citeseer2	434,102	16,036,720	37	4,457	5,946

framework with a custom implementation of HLDB in PostgreSQL and with $Neo4j$, a well-known, popular graph database.

We use the same network graphs as our previous work of [21] that are taken from the Stanford Large Network Dataset Collection [26] and the 10th Dimacs Implementation Challenge website [8]. All graphs are undirected, unweighted and strongly connected. We used collaboration graphs (DBLP, Citeseer1, Citeseer2) [24], social networks (Facebook [28], Slashdot1 and Slashdot2 [27]), networks with ground-truth communities (Amazon, Youtube) [31], web graphs (Notre Dame) [7] and location-based social networks (Gowalla) [11]. The graphs' average degree is between 3 and 37 and the PLL algorithm creates $26-4,457$ labels per vertex, requiring $0.03-5,946s$ for the hub labels' construction (see Table 8).

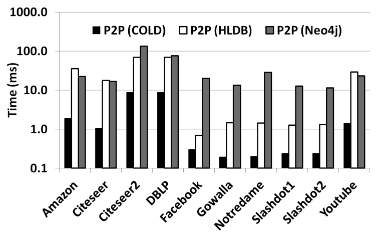
COLD and HLDB were implemented in PostgeSQL 9.3.6, 64bit with reasonable settings (8192Mb *shared buffers*, 64Mb *temp buffers*). We also used Neo4j Server v2.1.5. The Neo4j queries were formulated using *Cypher*, Neo4j's declarative query language and we report query times as they were returned by the server. Although Cypher may theoretically facilitate *one-to-many* queries (besides vertex-to-vertex), testing Neo4j with our datasets and the same number of target vertices we tested COLD with, resulted in a "`java.lang.Stack OverflowError`". Providing the server with additional resources[5] had no positive effect and thus there are no results for *one-to-many* queries and Neo4j.

We conducted experiments belonging to four query types: (i) *vertex-to-vertex*, (ii) kNN , (iii) RkNN and (iv) *one-to-many.* For each experiment, we used 10,000 random start vertices, reporting the average running time. Before each experiment, we restart the PostgreSQL and Neo4j servers for clearing their internal cache and we also clear the operating system's cache for accurate benchmarking. All charts are plotted in logarithmic scale.

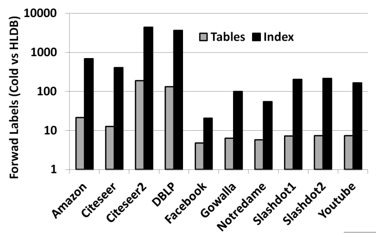
4.1 Performance on HDD

In our first round of experiments, we ran experiments on an HDD, specifically a SATA3 Seagate Barracude ST3000DM001 7200rpm with 64Mb cache.

[5] http://neo4j.com/developer/guide-performance-tuning/



(a) Vertex-to-vertex query times



(b) Memory size's difference between COLD and HLDB

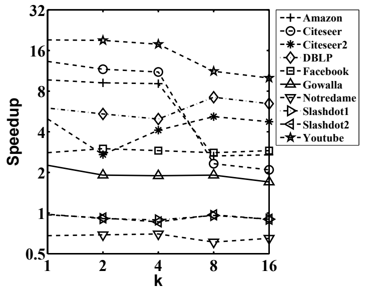
Fig. 2: Experiments on HDD for *vertex-to-vertex*

Vertex-to-vertex. Figure 2(a) shows results for vertex-to-vertex (v2v) queries for COLD, HLDB and Neo4j. Results show that COLD is consistently 2 - 20.7× faster than HLDB, with this difference amplified for the Citeseer1, Amazon and Youtube datasets (16.8, 19.1 and 20.7 respectively). Moreover, COLD is also 9 - 143× (for the *Gowalla* dataset) faster than Neo4j, which exhibits stable performance for all datasets, but is slower from both COLD and HLDB. For all datasets, COLD requires less than $9ms$ for answering v2v queries.

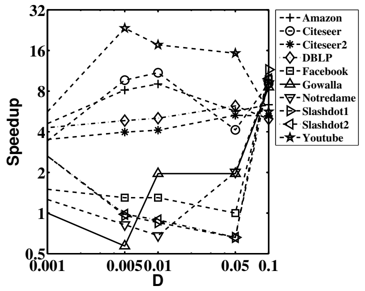
Figure 2(b) shows the difference in memory size for the DB tables *forcold* (COLD) and *forward* (HLDB) and their respective primary-key (PK) indexes. Results show that the size of the PK index in COLD is $3,600$ - $4,444$× smaller than for HLDB (for DBLP and Citeseer2 respectively). As expected, the difference in index sizes is almost identical to the $|HL|/|V|$ ratio, since *forcold* table has $|V|$ rows and *forward* has $|HL|$ rows. Likewise, the corresponding tables are 131 - 188× smaller for COLD. Thus, the techniques used for compressing the forward labels in COLD clearly achieve a considerable reduction in memory size, rendering our proposed framework suitable for real-world scenarios.

***k*NN.** Figure 3(a) shows the speedup of COLD compared to HLDB in the case of kNN queries for $D = 0.01$ and $k = \{1, 2, 4, 8, 16\}$. As described in Section 3.1, we have created two DB tables for each framework (COLD, HLDB), one for $kmax =\ 4$ and one for $kmax = 16$. Then the DB table for $kmax = 4$ is used for answering kNN queries for $k = 1$, $k = 2$ and $k = 4$ and the kNN table for $kmax = 16$ is used for answering kNN queries for $k = 8$ and $k = 16$. Results show that for $k = 1$, COLD is 5 - 19× faster for the five largest datasets (Amazon, Citeseer,Citeseer2, DBLP. Youtube) and although this speedup degrades for larger values of k, COLD remains consistently 2 - 10× faster even for $k = 16$. For the smaller datasets, performance between COLD and HLDB is quite similar, with COLD performing better on Facebook and Gowalla, while HLDB performs only marginally better for Slashdot1, Slashdot2 and Notredame. In all cases, COLD answers kNN queries for all datasets in less than $26ms$ even for $k = 16$.

In our second set of kNN experiments, we assess the performance of COLD vs HLDB for varying values of D. For each value for D, we have build separate



(a) kNN Speedup of COLD vs HLDB for $D = 0.01$ and varying values of k



(b) Speedup of COLD vs HLDB for $k = 4$ and varying values of D

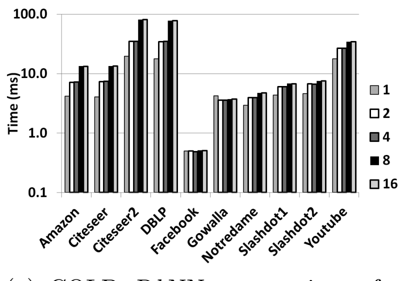
Fig. 3: kNN Experiments on HDD for COLD and HLDB

versions of *knntab* (HLDB) and *knncold* (COLD) DB tables for $D \cdot |V|$ objects selected at random from each dataset and $kmax = 4$. Figure 3(b) shows results for $k = 4$ and $D = \{0.001, 0.005, 0.01, 0.05, 0.1\}$. Again, for the five largest datasets COLD is consistently 3.4 - 23.4× faster than HLDB, whereas even for the smaller datasets, COLD is consistently 8.6 - 11.5× faster than HLDB for the largest value of D (for $D = 0.1$). Moreover, COLD may answer kNN queries for $k = 4$ on all datasets and all values of D in less than $14ms$.

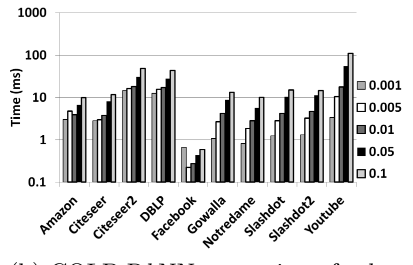
R*k*NN. For RkNN experiments, we only report COLD's performance, since there is no other SQL framework that supports these queries. In out first experiment, we report the performance of COLD for $D = 0.01$ and $k = \{1, 2, 4, 8, 16\}$. For all those queries we have built one version of the *knnres* DB table for $kmax = 16$ (see Section 3.1) and 3 separate *revcold* tables for $kmax = \{1, 4, 16\}$. As expected, for RkNN queries and $k = 1$ we use the *revcold* table built for $kmax = 1$, for $k = 2$, $k = 4$ we use the *revcold* table built for $kmax = 4$ and for $k = 8$, $k = 16$ we use the *revcold* table built for $kmax = 16$. Figure 4(a) presents the results. In all cases, COLD provides excellent query times that are below $20ms$ for $k = 1$ in all datasets and never exceed $82ms$ even for $k = 16$.

In our second set of RkNN experiments, we assess the performance of COLD for varying values of D. Figure 4(b) presents results for $k = 1$ (as this is the typical case for RkNN queries) and $D = \{0.001, 0.005, 0.01, 0.05, 0.1\}$. Results show that although COLD's performance degrades for larger values of D, RkNN query times are below $49ms$ for all datasets and values of D, with the exception of Youtube and $D = 0.1$ ($109.3ms$). Thus, COLD offers excellent and stable performance in RkNN queries for all all datasets and tested values of k and D.

One-to-Many. Again, COLD is the only SQL framework that supports *one-to-many queries*. Figure 5(a) presents the corresponding results for varying values of D ($D = \{0.001, 0.005, 0.01, 0.05, 0.1\}$). COLD answers such queries in less than a

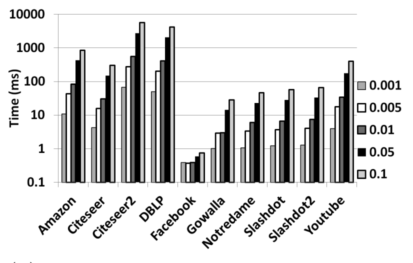


(a) COLD RkNN query times for $D = 0.01$ and varying values of k

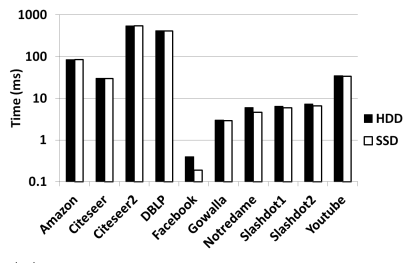


(b) COLD RkNN query times for $k = 1$ and varying values of D

Fig. 4: RkNN Experiments on HDD for COLD



(a) One-to-Many experiments for COLD varying values of D



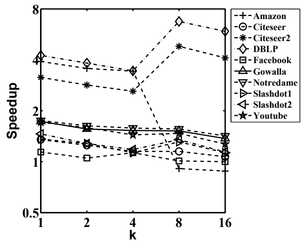
(b) COLD One-to-Many HDD vs SSD

Fig. 5: One-to-many experiments for COLD

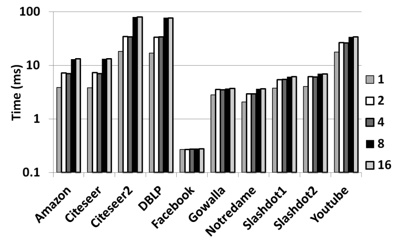
second for all datasets and values of D, except the Citeseer2 and DBLP datasets (those with the highest $|HL|/|V|$ ratio) that require $5601ms$ and $4170ms$ respectively, for $D = 0.1$. For such high values of D, the *one-to-many* query reaches the complexity of an *one-to-all* query and as expected, it cannot be any faster on a secondary storage device. Note that even specialized graph databases like Neo4j cannot support this type of queries for more than a 1,000 target objects, whereas *COLD answers one-to-many queries to 110,000 target objects in the Youtube dataset in* $401ms$ *with a simple SQL query.*

4.2 Performance on SSD

Having established the performance characteristics of COLD in the HDD, in our second round of experiments, we repeat some of the previous experiments, using a SSD to measure the impact of the secondary-storage device type to results. The SSD used is a SATA3 Crucial CT512MX100SSD1 MX100 512GB 2.5".



(a) kNN Speedup of COLD vs HLDB for $D = 0.01$ and varying values of k



(b) COLD RkNN query times for $D = 0.01$ and varying values of k

Fig. 7: kNN and RkNN SSD performance

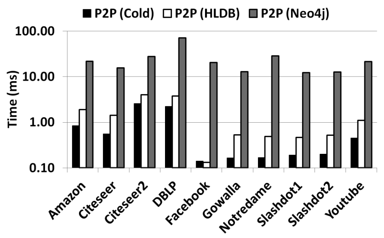


Fig. 6: SSD *vertex-to-vertex*

Vertex-to-vertex. Although the usage of SSD favors HLDB more than COLD (see Figure 6), COLD is consistently 1.6 - 3.2× faster than HLDB (except Facebook, the smallest of datasets). The SSD has almost no impact on Neo4j and thus, COLD is now 11-171× faster than $Neo4j$ on all datasets. Note, than on the SSD, COLD requires less than 0.9ms for all datasets and v2v queries, except the Citeseer2 and DBLP datasets (those with the highest $|HL|/|V|$ ratio). But even then, vertex-to-vertex queries still require less than 2.6ms for COLD.

***k*NN.** Figure 7(a) shows the performance speedup of COLD compared to HLDB in the case of kNN queries running on the SSD, for $D = 0.01$ and varying value of k. Again, although the SSD lowers the performance gap between COLD and HLDB, COLD is still faster on all datasets (except Facebook). In fact, COLD is 2.6 - 6.75× faster than HLDB for the high $|HL|/|V|$ ratio datasets (Citeseer2, HLDB) requiring less than 24.6ms even for $k = 16$.

R*k*NN. Figure 7(b) presents the results of the RkNN query time performance on COLD for $D = 0.01$ and varying value of k. Results show that SSD usage accelerates COLD by only 20% at most, which clearly demonstrates that COLD effectively minimized secondary storage utilization and thus adding a better secondary-storage medium provides minimal benefits for RkNN queries.

One-to-Many. Finally, Figure 5(b) compares *one-to-many* queries on HDD and SSD for COLD. Again, the SSD usage accelerates COLD by only 2- 30%, which further confirms the optimal secondary storage utilization of COLD.

4.3 Summary

Our experimentation has shown that our proposed COLD framework outperforms previous state-of-the-art HLDB in all performance benchmarks, including query performance, memory size and scalability. Using HDDs, COLD is $2-21\times$ faster for *vertex-to-vertex* queries and $5-19\times$ faster for kNN queries and the largest datasets. Using SSDs, COLD is $1.6-3.2\times$ faster than HLDB for *vertex-to-vertex* and up to $6.75\times$ faster for kNN queries. COLD also requires up to $4,444\times$ less storage space (indexes) and up to $188\times$ less storage space (DB tables) used for storing forward labels. Even specialized graph databases like Neo4j are outperformed by COLD, which is up to $143\times$ faster. Most importantly COLD may service additional (RkNN, one-to-many) queries, not handled by any other previous secondary-storage solutions, while providing excellent query times and optimal secondary-storage utilization even on standard hard drives.

5 Conclusions

This work presented COLD, a novel SQL framework for answering various exact distance queries for large-scale graphs on a database. Our results showed that COLD outperforms existing solutions (including specialized graph databases) on all levels, including query performance, secondary storage utilization and scalability. Moreover, COLD also answers RkNN and one-to-many queries, not handled by previous methods. This establishes COLD as a competitive database-driven framework for querying large-scale graphs. The paper gives the design and implementation details of COLD using a popular, open-source database system along with the actual SQL queries used in our implementation. This should allow for a simple replication of our results and encourage other researchers to expand the COLD framework towards handling more complex queries and test-cases.

Acknowledgements

This work was partially supported by EU (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund and the EU/Greece funded KRIPIS Action: MEDA Project. D. Pfoser's work was partially supported by the NGA NURI grant HM02101410004.

References

1. I. Abraham, D. Delling, A. Fiat, A. V. Goldberg, and R. F. Werneck. Hldb: Location-based services in databases. In *SIGSPATIAL GIS*. ACM, November 2012.
2. I. Abraham, D. Delling, A. Goldberg, and R. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In P. Pardalos and S. Rebennack, editors, *Experimental Algorithms*, volume 6630 of *Lecture Notes in Computer Science*, pages 230–241. Springer Berlin Heidelberg, 2011.

3. I. Abraham, D. Delling, A. Goldberg, and R. Werneck. Hierarchical hub labelings for shortest paths. In L. Epstein and P. Ferragina, editors, *Algorithms – ESA 2012*, volume 7501 of *Lecture Notes in Computer Science*, pages 24–35. Springer Berlin Heidelberg, 2012.
4. T. Akiba, Y. Iwata, K. Kawarabayashi, and Y. Kawata. Fast shortest-path distance queries on road networks by pruned highway labeling. In *2014 Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2014, Portland, Oregon, USA, January 5, 2014*, pages 147–154, 2014.
5. T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, USA*, pages 349–360, 2013.
6. T. Akiba, Y. Iwata, and Y. Yoshida. Pruned landmark labeling [online]. `https://github.com/iwiwi/pruned-landmark-labeling`, 2015.
7. R. Albert, H. Jeong, and A.-L. Barabási. The diameter of the world wide web. *CoRR*, cond-mat/9907038, 1999.
8. D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, editors. *Graph Partitioning and Graph Clustering - 10th DIMACS Implementation Challenge Workshop, Georgia Institute of Technology, Atlanta, GA, USA, February 13-14, 2012. Proceedings*, volume 588 of *Contemporary Mathematics*. American Mathematical Society, 2013.
9. H. Bast, D. Delling, A. V. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. *CoRR*, abs/1504.05140, 2015.
10. F. Borutta, M. A. Nascimento, J. Niedermayer, and P. Kröger. Monochromatic rknn queries in time-dependent road networks. In *Proceedings of the Third ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, MobiGIS '14, pages 26–33, New York, NY, USA, 2014. ACM.
11. E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 1082–1090, 2011.
12. E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and distance queries via 2-hop labels. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, pages 937–946, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.
13. D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Customizable route planning. In *Proceedings of the 10th international conference on Experimental algorithms*, SEA'11, pages 376–387, Berlin, Heidelberg, 2011.
14. D. Delling, A. V. Goldberg, T. Pajor, and R. F. Werneck. Robust distance queries on massive networks. In *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, pages 321–333, 2014.
15. D. Delling, A. V. Goldberg, and R. F. Werneck. Hub label compression. In *Experimental Algorithms, 12th International Symposium, SEA 2013, Rome, Italy, June 5-7, 2013. Proceedings*, pages 18–29, 2013.
16. D. Delling, A. V. Goldberg, and R. F. F. Werneck. Faster batched shortest paths in road networks. In *ATMOS*, pages 52–63, 2011.
17. D. Delling and R. F. Werneck. Customizable point-of-interest queries in road networks. In *21st SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2013, Orlando, FL, USA, November 5-8, 2013*, pages 490–493, 2013.

18. D. Delling and R. F. F. Werneck. Better bounds for graph bisection. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 407–418, 2012.
19. A. Efentakis and D. Pfoser. Optimizing landmark-based routing and preprocessing. In *CTS 2013, 6th ACM SIGSPATIAL International Workshop on Computational Transportation Science, November 5, 2013, Orlando, FL, USA*, page 25, 2013.
20. A. Efentakis and D. Pfoser. GRASP. Extending graph separators for the single-source shortest-path problem. In A. S. Schulz and D. Wagner, editors, *Algorithms - ESA 2014*, volume 8737 of *Lecture Notes in Computer Science*, pages 358–370. Springer Berlin Heidelberg, 2014.
21. A. Efentakis and D. Pfoser. ReHub. Extending hub labels for reverse k-nearest neighbor queries on large-scale networks. *arXiv preprint arXiv:1504.01497*, 2015.
22. A. Efentakis, D. Pfoser, and Y. Vassiliou. SALT. A unified framework for all shortest-path query variants on road networks. In E. Bampis, editor, *Experimental Algorithms*, volume 9125 of *Lecture Notes in Computer Science (To appear)*. Springer International Publishing, 2015.
23. C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 210–219, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
24. R. Geisberger, P. Sanders, and D. Schultes. Better approximation of betweenness centrality. In J. I. Munro and D. Wagner, editors, *ALENEX*, pages 90–100. SIAM, 2008.
25. M. Jiang, A. W. Fu, R. C. Wong, and Y. Xu. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *PVLDB*, 7(12):1203–1214, 2014.
26. J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.
27. J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.
28. J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 548–556, 2012.
29. PostgreSQL. The world's most advanced open source database [online]. `http://www.postgresql.org/`, 2015.
30. M. Safar, D. Ibrahimi, and D. Taniar. Voronoi-based reverse nearest neighbor query processing on spatial networks. *Multimedia Systems*, 15(5):295–308, 2009.
31. J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 745–754, 2012.
32. M. L. Yiu, D. Papadias, N. Mamoulis, and Y. Tao. Reverse nearest neighbors in large graphs. *Knowledge and Data Engineering, IEEE Transactions on*, 18(4):540–553, April 2006.
33. R. Zhong, G. Li, K.-L. Tan, and L. Zhou. G-tree: An efficient index for knn search on road networks. In *Proceedings of the 22nd ACM International Conference on Conference on Information Knowledge Management*, CIKM '13, pages 39–48, New York, NY, USA, 2013. ACM.

RDF Keyword Search based on Keywords-To-SPARQL Translation

Katerina Gkirtzou
Athena Research Center
Marousi, Greece
katerina.gkirtzou@imis.athena-innovation.gr

George Papastefanatos
Athena Research Center
Marousi, Greece
gpapas@imis.athena-innovation.gr

Theodore Dalamagas
Athena Research Center, GR
Marousi, Greece
dalamag@imis.athena-innovation.gr

ABSTRACT
In this paper, we present a summary of our work on RDF keyword search. Given a set of keywords, our method automatically generates a set of candidate SPARQL queries, and their natural language description, to be evaluated on the RDF data graph. We discuss our approach, highlighting current and future directions.

1. INTRODUCTION
Linked Data is the most common practice for publishing, sharing and managing information in the Data Web, and is implemented with the RDF 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. SPARQL is the de facto query language for RDF. However, forming SPARQL queries requires both knowing the SPARQL syntax, as well as knowing the vocabulary used to describe the data. For this reason, keyword-based search has been proposed, allowing an intuitive way for searching an RDF dataset.

In this short paper, we present a summary of our previous work in [1, 2], highlighting current and future directions. Specifically, we present our 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, automatically generates a set of candidate SPARQL queries, i.e., SPARQL queries that try to capture users information need as expressed by the keywords used. To further assist our collaborators in the evaluation process, we provide also a natural language description of the generated SPARQL queries. To achieve this, we exploit the SPARQL2NL [3] engine which allows the verbalization of SPARQL queries into natural language. A fully functionable prototype is available at:

http://snf-624527.vm.okeanos.grnet.gr:8080/KeywordSearchDiana/web/

2. METHOD
Briefly, given a set of *n* keywords, we perform the following steps in order to generate candidate SPARQL queries:

1. For each keyword k_i, we retrieve all its matches M_i on the RDF data graph.
2. We calculate all possible combinations $C = M_1 \times M_2 \times \cdots \times M_n = \{ c = (m_1, \cdots, m_n) | m_i \in M_i \ \forall i = 1, \cdots, n\}$ of all the matched elements M_i where $i = 1, \cdots, n$.
3. For each combination $c \in C$ that contains one matched element m_i per keyword k_i, we create an *augmented summary graph* G_c.
4. From each augmented summary graph G_c, we generate the *query pattern graph* G_c^{QP}.
5. We translate each query pattern graph G_c^{QP}. into a SPARQL query.

Next, we elaborate on the details. Consider the biological RDF dataset shown in Figure 1. The dataset is depicted as an RDF data graph, where oval shape vertices represent RDF entities, diamond shape vertices represent RDF classes and rectangle shape vertices represent literals. Similarly, dashed edges represent entity-to-attribute properties, while solid ones represent inter-entity properties. Let us assume that the user has provided the keywords *MIMAT0000251, name* and *hasTarget*. The first step is to match the keywords to elements in the RDF data graph:

1. *MIMAT0000251* matches to the literal "MIMAT0000251" that is connected via the property "accession" with an RDF entity of "Mature" type,
2. *name* matches to the literal "NAME" that is connected via the property "change" met with RDF entity of type "Mature" and to the entity-to-attribute property "name" met with entities of type "Hairpin", "Mature", "Species" and "Gene", resulting in 5 possible matches, and
3. *hasTarget* matches to the inter-entity property "hasTarget" met with subject of type "Interaction" and object of type "Transcript".

The second step is to calculate all possible combinations of the matched elements and for each combination *c* create the augmented summary graph G_c, extrapolate the query pattern graph G_c^{QP} and map it to a SPARQL query. In this example, there are 5 possible combinations. Let us examine one combination, where the *name* keyword matches to the literal "NAME".

Augmented symmary graph. The augmented summary graph G_c is a combination of an aggregated representation of the RDF data graph G, enriched with graph elements for each matched element $m_i, i = 1, \cdots, n$. More specifically, all RDF entities from the RDF data graph G that have the same type of RDF class are represented by a vertex labelled with the name of the RDF class.

Similarly, all inter-entity properties of the same type are represented by a directed edge between the aggregated vertex representation of the subjects and the aggregated vertex representation of the objects. The edge is also labelled with the property's name. Note that entity-to-attribute properties as well as literal values are omitted from the summary representation. Overall, the augmented graph is actually an abstraction of the RDF data graph G. The augmented summary graph G_c contains also graph elements for each element m_i from the set of matched elements c. More specifically, if the matched element m_i is a literal value, then the graph is extended by a directed edge and a vertex. The edge represents the entity-to-attribute property that the matched element is met with in the RDF data graph, while the vertex is the matched element m_i itself. Note that the edge is attached from the aggregated vertex representation of the subject to the newly inserted vertex. Similarly, if the matched element m_i is an entity-to-attribute property, then the graph is extended by a directed edge and a vertex. The edge represents the entity-to-attribute property, i.e. the matched element m_i, and it is attached from the aggregated vertex representation of the subject to the newly inserted vertex. The difference from the previous case is that the latter vertex represents the unknown literal of the property. Note that if the same entity-to-attribute property is met with multiple RDF entities of different RDF types in the RDF data graph that would lead to different sets c. An example of the Augmented Summary Graph for the combination under investigation is shown in Figure 2.

Query pattern graph. In order to extract the query pattern graph G_c^{QP} from the augmented summary graph G_c, we calculate the shortest paths between every pair of matched elements and we combine all of them into one connected subgraph. 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_c.

For the Augmented Summary Graph of Figure 2, since we have three keywords, we need to calculate three shortest paths, which are the following ones:

1. MIMAT0000251 → accession → Mature → hasMature → Interaction → hasTarget,
2. MIMAT0000251 → accession → Mature → change → NAME and
3. NAME → change → Mature → hasMature → Interaction → hasTarget.

We then combine the shortest paths into a single connected component, resulting to the query pattern graph shown in Figure 3. 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 any of the shortest paths.

Candidate SPARQL generation. The final step of the mapping process is to translate the query pattern graph G_c^{QP} into a SPARQL query. Note that the vertices of the query pattern graph G_c^{QP} are either known or unknown literal values and aggregated representations. We need to connect the latter type of vertices and the vertices of unknown literal values with variables in order to form the SPARQL triple patterns. Note that labels of the vertices can be used as constants in the triple patterns, while the labels of the edges as predicates. To produce conjunctive SPARQL queries, given the above observations for every vertex $v \in G_c^{QP}$, we perform the following:

- if v is a literal, do nothing
- if v is an unknown literal, then connect the vertex into a new variable $var(v)$.
- If v is a aggregated representation for entities of RDF type *class* with label $label(v) = class$, then the vertex is connected into a new variable $var(v)$ and produce the following SPARQL triple $var(v)$ `rdf:type` $label(v)$.

Similarly, for every edge $e \in G_c^{QP}$:

- If e represents an inter-entity property between a vertex $subject$ and a vertex $object$, then we produce the triple pattern:
  $var(subject)\ label(e)\ var(object)$.
- If e represents an entity-to-attribute property between a vertex $subject$ and a vertex $object$ that is a literal, then we produce the triple pattern:
  $var(subject)\ label(e)\ label(object)$.

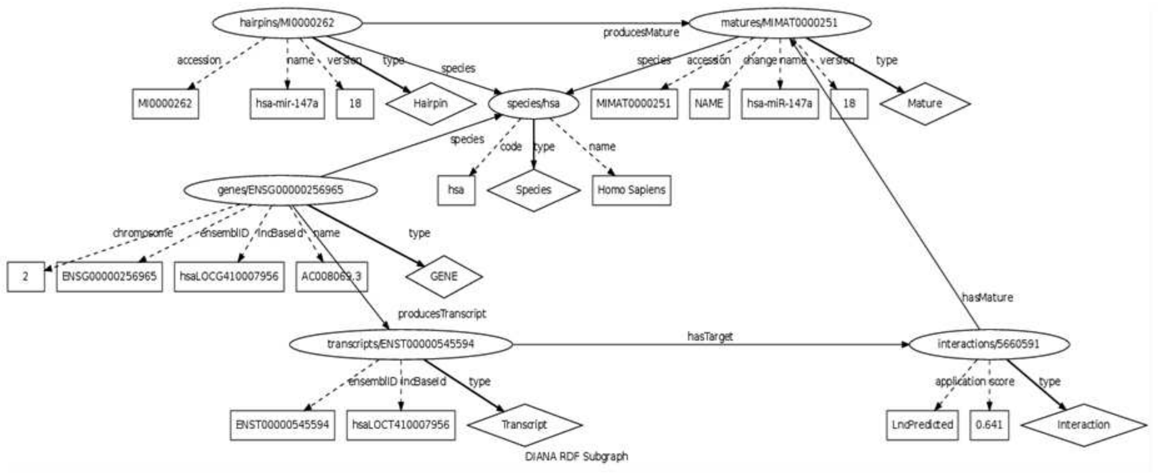


Figure 1. Example of an RDF data graph

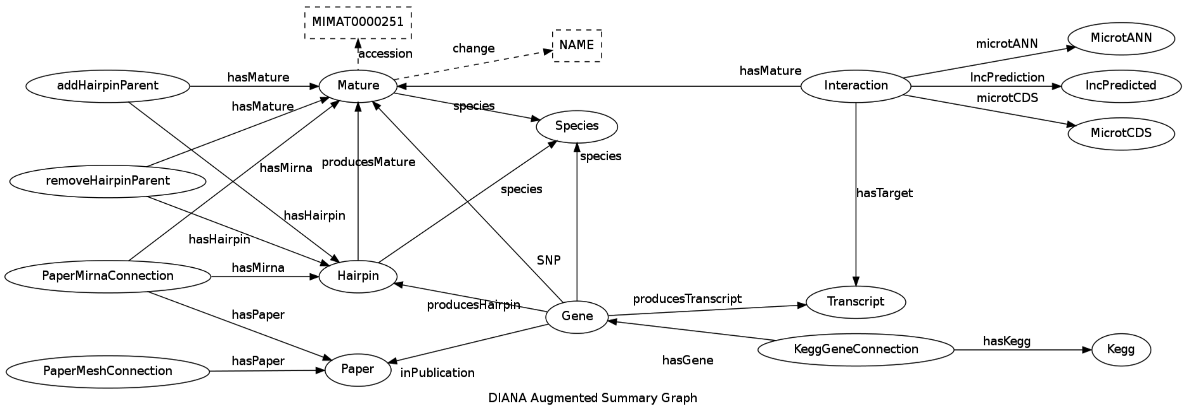


Figure 2. Example of an augmented summary graph

- If e represents an entity-to-attribute property between a vertex $subject$ and a vertex $object$ that is an unknown literal, then we produce the triple pattern

  $var(subject)\ label(e)\ var(object)$.

In our example, the pattern graph of Figure 3 is mapped to the following SPARQL query:

```
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 "MIMAT0000251". ?M diana:change "NAME".}
```

3. DISCUSSION AND CONCLUSIONS

The keyword search problem over graph structured data has been widely explored [4,5,7]. These works follow three basic steps: a) mapping the keyword elements to structured data elements b) connect the keyword elements by searching for substructures on the data, and c) return as output the retrieved substructures, given a scoring function. Our method follows the approach in [6], where, instead of computing the answers directly on the data, they produce structured queries from RDF summary graphs [6, 8]. Compared to [6], we do not focus on top-k queries. Also, we create multiple augmented graphs, and use the notion of shortest paths to create a query pattern graph.

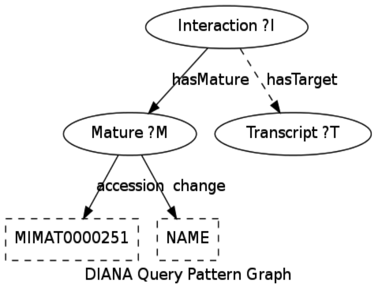


Figure 3. Example of a query pattern graph

We have performed [1] a preliminary evaluation of our method. We asked domain experts to provide keyword queries for the RDF dataset of Figure 1, along with natural language descriptions. Then, we checked whether the generated SPARQL queries and the natural language descriptions are close to the descriptions provided. Results [2] are promising, since most of the generated descriptions match those of users'. We plan to work further on the following directions: (a) investigate alternative ways for building query pattern graphs, (b) investigate search diversification semantics, (c) adopt string matching methods for keyword-to-node matching, and (d) evaluate the performance of data structures used to make the keyword-to-SPARQL translation.

4. ACKNOWLEDGEMENTS

Work supported by the LODGOV project, Research Program. ARISTEIA, GSRT/Ministry of Education - European Regional Development Fund, and the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework - Research Funding Program THALES.

5. REFERENCES

[1] K. Gkirtzou, K. Karozos, V. Vassalos, and T. Dalamagas. *Keywords-to-sparql translation for rdf data search and exploration*. In Proceedings of TPDL'15, Poznań, Poland, Sep 14-18, 2015 (to appear).

[2] M. Meimaris, G. Alexiou, K. Gkirtzou, G. Papastefanatos, and T. Dalamagas. *RDF resource search and exploration with LinkZoo*. In Proceedings of DATA'15, Colmar, Alsace, France, July 20-22, 2015.

[3] A. C. Unger, J. Lehmann, D. Gerber. *Sorry, I Don'T Speak SPARQL: Translating SPARQL Queries into Natural Language*. In Proceedings of WWW'13, Rio de Janeiro, Brazil, May 13 - 17, 2013.

[4] H. He, H. Wang, J. Yang, and P. S. Yu. *BLINKS: Ranked Keyword Searches on Graphs*. In Proceedings of SIGMOD'07, Beijing, China, June 11-14, 2007.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti and S. Sudarshan. *Keyword searching and browsing in databases using BANKS*. In Proceedings of ICDE'02, San Jose , CA, USA, Feb 26 - Mar 1, 2002.

[6] D. Tran, H. Wang, S. Rudolph and P. Cimiano. *Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data*. In Proceedings of ICDE'09, Shanghai, China, Mar 29 - Apr 2, 2009.

[7] K. Stefanidis and I. Fundulaki. Keyword *Search on RDF Graphs: It Is More Than Just Searching for Keywords*. In proceedings of ESWC'15, Portoroz, Slovenia, May 31 - Jun 4, 2015.

[8] G. Troullinou, H. Kondylakis, E. Daskalaki and D. Plexousakis. *RDF Digest: Efficient Summarization of RDF/S KBs*. In proceedings of ESWC'15, Portoroz, Slovenia, May 31 - Jun 4, 2015.

Towards Scalable Visual Exploration of Very Large RDF Graphs

Nikos Bikakis[1,2], John Liagouris[3], Maria Kromida[1], George Papastefanatos[2], and Timos Sellis[4]

[1] NTU Athens, Greece [2] ATHENA Research Center, Greece
[3] ETH Zürich, Switzerland [4] RMIT University, Australia

Abstract. In this paper, we outline our work on developing a disk-based infrastructure for efficient visualization and graph exploration operations over very large graphs. The proposed platform, called graphVizdb, is based on a novel technique for indexing and storing the graph. Particularly, the graph layout is indexed with a spatial data structure, i.e., an R-tree, and stored in a database. In runtime, user operations are translated into efficient spatial operations (i.e., window queries) in the backend.

Keywords: graphVizdb, graph data, disk based visualization tool, RDF graph visualization, spatial, visualizing linked data, partition based graph layout.

1 Introduction

Graph visualization and exploration is a core task in various application domains, such as scientific data management, social networks and the Data Web. The wide availability of vast amounts of graph-structured data, RDF in the case of the Data Web, demands for user-friendly methods and tools for data exploration and knowledge uptake. We consider some core challenges related on the management and visualization of very large RDF graphs; e.g., the Wikidata RDF graph has more than 300M nodes and edges.

First, their size exceeds the capabilities of memory-based layout techniques and libraries, enforcing disk-based implementations. Then, graph rendering is a time consuming process; even drawing a small part of the graph (containing a few hundreds of nodes) requires considerable time when we assume real-time systems. The same holds for graph interaction and navigation. Most operations, such as zoom in/out and move, are not easily implemented to large dense graphs, as their implementations require redrawing and re-layout large parts of them.

Related works in the field handle very large graphs through hierarchical visualization approaches. Although hierarchical approaches provide fancy visualizations with low memory requirements, their applicability is heavily based on the particular characteristics of the input dataset. In most cases, the hierarchy is constructed by exploiting clustering and partitioning methods [?,?,?,?,?]. In other works, the hierarchy is defined with hub-based [?] and destiny-based [?]

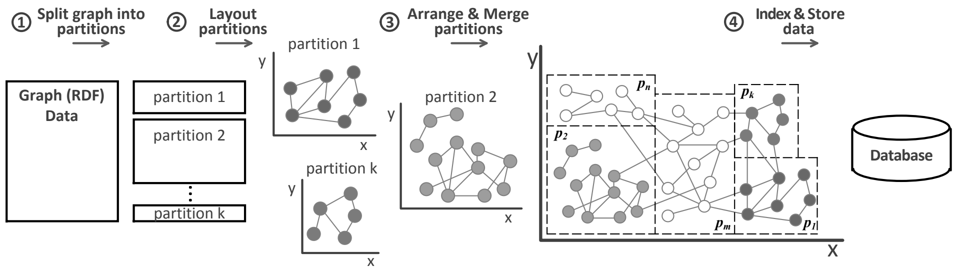


Fig. 1. Preprocessing Overview

techniques. [?] supports ad-hoc hierarchies which are manually defined by the users. Some of these systems offer a disk-based implementation [?,?,?] whereas others keep the whole graph in main memory [?,?,?,?,?].

In the context of the Web of Data [?,?,?,?,?,?,?], there is a large number of tools that visualize RDF graphs (adopting a node-link approach); the most notable ones are *ZoomRDF* [?], *Fenfire* [?], *LODWheel* [?], *RelFinder* [?] and *LODeX* [?]. All these tools require the whole graph to be loaded on the UI. Several tools that follow the same non-scalable approach have also been developed in the field of ontology visualization [?,?].

In contrast to all existing works, we introduce a generic platform, called graphVizdb, for scalable graph visualization that do not necessarily depend on the characteristics of the dataset. The efficiency of the proposed platform is based on a novel technique for indexing and storing the graph. The core idea is that in a preprocessing phase, the graph is drawn, using any of the existing graph layout algorithms. After drawing the graph, the coordinates assigned to its nodes (with respect to a Euclidean plane) are indexed with a spatial data structure, i.e., an R-tree, and stored in a database. In runtime, while the user is navigating over the graph, based on the coordinates, specific parts of the graph are retrieved and send to the user.

2 Platform Overview

The graphVizdb platform is built on top of two main concepts: (1) it is based on a "spatial-oriented" approach for graph visualization, similar to approaches followed in browsing maps; and (2) it adopts a disk-based implementation for supporting interaction with the graph, i.e., a database backend is used to index and store graph and visual information.

Partition-based graph layout. Here we outline the partition-based approach adopted by the graphVizdb in order to handle very large graph. Recall that, for graph layout, the graph is drawn once in a preprocessing phase, using any of the existing graph layout algorithms. However several graph layout algorithms require large amount of memory in order to draw very large graphs. In order to overcome this problem, our partition-based approach (outlined in Figure 1) is described next.

(1) Initially, the graph (RDF) data is divided into a set of smaller sub-graphs (i.e., partitions) using a graph partitioning algorithm. At the same time, the graph partitioning algorithm tries to minimize the number of edges connecting nodes in different partitions. (2) Then, using a graph layout algorithm, each of the sub-graph resulted from the graph partitioning, is visualized into a Euclidean plane, excluding (i.e., not visualizing) the edges connecting nodes through different partitions (i.e., crossing edges). (3) The visualized partitions are organized and combined into a "global" plane using a greedy algorithm whose goal is twofold. First, it ensures that the distinct sub-graphs do not overlap on the plane, and at the same time it tries to minimize the total length of the crossing edges. (4) Based on the "global" plane, the coordinates for each node and edge are indexed and stored in the database.

Spatial operations for graph exploration. In graphVizdb, most of the user's requests are translated into simple spatial operations evaluated over the database. In this context, *window queries* (i.e., spatial range queries that retrieve the information contained with in a specific spatial region) are the core operation for most user's requests. The user navigates on the graph by moving the viewing window. When the window is moved, its new coordinates with respect to the whole canvas are tracked on the client side, and a window query is sent to the server. The query is evaluated on the server using the R-tree indexes. This way, for each user request, graphVizdb efficiently renders only visible parts of the graph, minimizing in this way both backend-frontend communication cost as well as rendering and layout time. Additionally, more sophisticated operations, e.g., abstraction/enrichment zoom operations are also implemented using spatial operations.

Implementation. We have implemented a graphVizdb prototype[1] which provides interactive visualization over large graphs. The prototype offers three main operations: (1) interactive navigation, (2) multi-level exploration, and (3) keyword search. We use MySQL for data storing and indexing, the Jena framework for RDF data handling, Metis[2] for graph partitioning, and Graphviz[3] for drawing the graph partitions. In the front-end, we use mxgraph[4], a client-side JavaScript visualization library. A video presenting the basic functionality of our prototype is available at: `vimeo.com/117547871`.

References

1. J. Abello, F. van Ham, and N. Krishnan. ASK-GraphView: A Large Scale Graph Visualization System. *IEEE Trans. Vis. Comput. Graph.*, 12(5), 2006.

[1] graphvizdb.imis.athena-innovation.gr
[2] glaros.dtc.umn.edu/gkhome/views/metis
[3] www.graphviz.org
[4] www.jgraph.com

2. M. Alonen, T. Kauppinen, O. Suominen, and E. Hyvönen. Exploring the linked university data with visualization tools. In *ESWC*, 2013.
3. D. Archambault, T. Munzner, and D. Auber. GrouseFlocks: Steerable Exploration of Graph Hierarchy Space. *IEEE Trans. Vis. Comput. Graph.*, 14(4), 2008.
4. D. Auber. Tulip - A Huge Graph Visualization Framework. In *Graph Drawing Software*. 2004.
5. M. Bastian, S. Heymann, and M. Jacomy. Gephi: An Open Source Software for Exploring and Manipulating Networks. In *ICWSM*, 2009.
6. F. Benedetti, L. Po, and S. Bergamaschi. A Visual Summary for Linked Open Data sources. In *ISWC*, 2014.
7. N. Bikakis, M. Skourla, and G. Papastefanatos. rdf:SynopsViz - A Framework for Hierarchical Linked Data Visual Exploration and Analysis. In *ESWC*, 2014.
8. J. M. Brunetti, S. Auer, R. García, J. Klímek, and M. Necaský. Formal Linked Data Visualization Model. In *IIWAS*, 2013.
9. A. Dadzie and M. Rowe. Approaches to visualising Linked Data: A survey. *Semantic Web*, 2(2), 2011.
10. M. Dudás, O. Zamazal, and V. Svátek. Roadmapping and Navigating in the Ontology Visualization Landscape. In *EKAW*, 2014.
11. B. Fu, N. F. Noy, and M.-A. Storey. Eye Tracking the User Experience - An Evaluation of Ontology Visualization Techniques. *Semantic Web Journal*, 2015.
12. T. Hastrup, R. Cyganiak, and U. Bojars. Browsing Linked Data with Fenfire. In *WWW*, 2008.
13. P. Heim, S. Lohmann, and T. Stegemann. Interactive Relationship Discovery via the Semantic Web. In *ESWC*, 2010.
14. J. F. R. Jr., H. Tong, A. J. M. Traina, C. Faloutsos, and J. Leskovec. GMine: A System for Scalable, Interactive Graph Visualization and Mining. In *VLDB*, 2006.
15. Z. Lin, N. Cao, H. Tong, F. Wang, U. Kang, and D. H. P. Chau. Demonstrating Interactive Multi-resolution Large Graph Exploration. In *ICDM*, 2013.
16. S. Mazumdar, D. Petrelli, and F. Ciravegna. Exploring user and system requirements of linked data visualization through a visual dashboard approach. *Semantic Web*, 5(3), 2014.
17. S. Mazumdar, D. Petrelli, K. Elbedweihy, V. Lanfranchi, and F. Ciravegna. Affective graphs: The visual appeal of Linked Data. *Semantic Web*, 6(3), 2015.
18. M. Stuhr, D. Roman, and D. Norheim. LODWheel - JavaScript-based Visualization of RDF Data. In *Workshop on Consuming Linked Data*, 2011.
19. C. Tominski, J. Abello, and H. Schumann. CGV - An interactive graph visualization system. *Computers & Graphics*, 33(6), 2009.
20. L. D. Vocht, A. Dimou, J. Breuer, M. V. Compernolle, R. Verborgh, E. Mannens, P. Mechant, and R. V. de Walle. A visual exploration workflow as enabler for the exploitation of linked open data. In *IESD*, 2014.
21. K. Zhang, H. Wang, D. T. Tran, and Y. Yu. ZoomRDF: semantic fisheye zooming on RDF data. In *WWW*, 2010.
22. M. Zinsmaier, U. Brandes, O. Deussen, and H. Strobelt. Interactive Level-of-Detail Rendering of Large Graphs. *IEEE Trans. Vis. Comput. Graph.*, 18(12), 2012.

Acknowledgement. This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Programs "Education and Lifelong Learning" - Funding Program: THALIS and "Competitiveness and Entrepreneurship" (OPCE II) - Funding Program: KRIPIS of the National Strategic Reference Framework (NSRF).

rdf:SynopsViz – A Framework for Hierarchical Linked Data Visual Exploration and Analysis

Nikos Bikakis[1,2] Melina Skourla[1] George Papastefanatos[2]

[1]National Technical University of Athens, Greece
[2]IMIS, ATHENA Research Center, Greece

Abstract. The purpose of data visualization is to offer intuitive ways for information perception and manipulation, especially for non-expert users. The Web of Data has realized the availability of a huge amount of datasets. However, the volume and heterogeneity of available information make it difficult for humans to manually explore and analyse large datasets. In this paper, we present rdf:SynopsViz, a tool for hierarchical charting and visual exploration of Linked Open Data (LOD). Hierarchical LOD exploration is based on the creation of multiple levels of hierarchically related groups of resources based on the values of one or more properties. The adopted hierarchical model provides effective information abstraction and summarization. Also, it allows efficient -on the fly- statistic computations, using aggregations over the hierarchy levels.

Keywords: Visual analytics, Semantic Web, LOD, RDF visualization, Data exploration, RDF Statistics, RDF Charts, Faceted search, RDF Facets.

1 Introduction

The purpose of data visualization is to offer intuitive ways for information perception and manipulation that essentially amplify, especially for non-expert users, the overall cognitive performance of information processing. This is of great importance in the Web of Data, where the volume and heterogeneity of available information make difficult for humans to manually explore and analyse large datasets. An important challenge is that visualization techniques must offer scalability and efficient processing for on the fly visualization of large datasets. They must also employ appropriate data abstractions and aggregations for avoiding information overloading due to the size and diversity of the data presented to the user. Finally, they must be generic and provide uniform and intuitive visualization results across multiple domains.

In this work, we present rdf:SynopsViz, a framework for hierarchical charting and exploration of Linked Open Data (LOD). Hierarchical LOD exploration realized through the creation of multiple levels of hierarchically related groups of resources based on the values of one or more properties. For example, a numerical group, characterized by a numerical range, comprises all resources with a property value within the range of this group. Hierarchical browsing can address the problem of information overloading as it provides information abstraction and summarization [1]. It can also offer rich insights on the underlying data when combined with rich statistical information on the groups and their contents.

The key features of rdf:SynopsViz framework are summarized as follows: (1) It adopts a *hierarchical model* for RDF data visualization, browsing and analysis. (2) It offers *automatic* on-the-fly hierarchy construction based on data distribution, as well as *user-defined* hierarchy construction based on user's preferences. (3) Provides *faceted* browsing and filtering over classes and properties. (4) Integrates *statistics with visualization*; visualizations have been enriched with useful statistics and data information. (5) Offers several visualizations techniques (e.g., timeline, chart, treemap). (6) Provides a large number of dataset's *statistics* regarding the: data-level (e.g., number of sameAs triples), schema-level (e.g., most common classes/properties), and structure level (e.g., entities with the larger in-degree). (7) Provides numerous *metadata* related to the dataset: licensing, provenance, linking, availability, undesirability, etc. The latter are useful for assessing data quality [13].

2 Framework Overview

The architecture of rdf:SynopsViz is presented in Figure 1. Our scenario involves three main parts: the Client GUI, the rdf:SynopsViz framework, and the input data. The *Client* part, corresponds to the framework's front-end offering several functionalities to the end-users (e.g., statistical analysis, facet search, etc.). rdf:SynopsViz consumes RDF data as *Input data*; optionally, OWL-RDF/S vocabularies/ontologies describing the input data can be loaded. Next, we describe the basic components of the rdf:SynopsViz framework.

In the preprocessing phase, the *Data and Schema Handler* parses the input data and inferes schema information (e.g., properties domain(s)/range(s), class/property hierarchy, type of instances, type of properties, etc.). *Facets Generator* generates class and property facets over input data. *Statistics Generator* computes several statistics regarding the schema, instances and graph structure of the input dataset, such as the number of different types of classes and properties, or the number of sameAs triples, or finally the average in/out degree of the RDF graph, respectively. *Metadata Extractor* collects dataset metadata which can be used for data quality assessment. *Hierarchical Model Module* adopts our

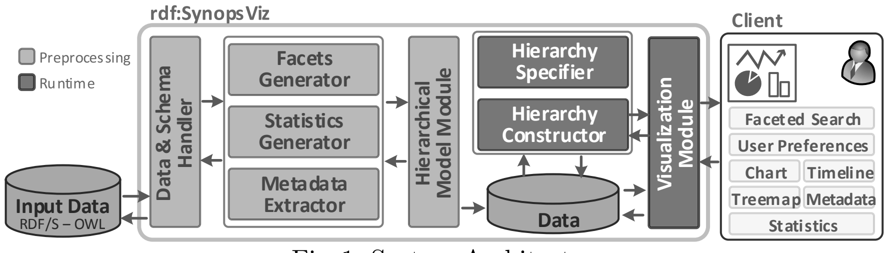


Fig. 1: System Architecture

hierarchy model and stores the initial data enriched with the information computed during the preprocessing phase.

During runtime the following components are involved. *Hierarchy Specifier* is responsible for managing the configuration parameters of our hierarchy model, e.g., the number of hierarchy levels, the number of nodes per level, and providing this information to the Hierarchy Constructor. *Hierarchy Constructor* implements the hierarchy model. Based on the selected facets, and the hierarchy configuration: it determines the hierarchy of groups and the contained triples, and computes the statistics about their contents (e.g., range, variance, mean, number of triples contained, etc.). *Visualization Module* allows the interaction between the user and the framework, allowing several operations (e.g, navigation, filtering, hierarchy specification) over the visualized data.

3 Implementation & Demonstration Outline

Implementation. rdf:SynopsViz is implemented on top of several open source tools and libraries. Regarding visualization libraries, we use Highcharts[1], for the area and timeline charts. and Google Charts[2] for treemap and pie charts. Additionally, it uses Jena framework[3] for RDF data handing and Jena TDB for RDF storing.

The web-based prototype of rdf:SynopsViz is available at `83.212.125.131:8084/synopsViz`. Also a video demonstrating the scenario presented below is available at `http://youtu.be/8v-He1U4oxs`.

Demonstration scenario. First, the attenders will be able to select a dataset from a number of offered real-word datasets (e.g., dbpedia, Eurostat, World Bank, U.S. Census, etc.) or upload their own. Then, for the selected dataset, the attendees are able to examine several of the dataset's *metadata*, and explore several datasets's *statistics*.

Using the facets panel, the attenders are able to navigate and filter data based on classes, numeric and date properties. In addition, through facets navigation several information about the classes and properties (e.g., number of instances, domain(s), range(s), IRI, etc.) are provided to the users through the UI.

The attenders are able to navigate over data by considering properties' values. Particularly, area charts and timeline-area charts are used to visualize the resources considering the user's selected properties. Classes' facets can also be used to filter the visualized data. Initially, the top level of the hierarchy is presented providing an overview of the data, organized into top-level groups; the user can interactively zoom in and out the group of interest, up to the actual values of the raw input data. At the same time, statistical information concerning the hierarchy groups as well as their contents (e.g., mean value, variance, sample data, etc.) are presented.

[1] www.highcharts.com
[2] developers.google.com/chart
[3] jena.apache.org

In addition, the attenders are able to navigate over data, through class hierarchy. Selecting one or more classes, the attenders can interactively navigate over the class hierarchy using treemaps. In rdf:SynopsViz the treemap visualization has been enriched with schema and statistical information. For each class, schema metadata (e.g., number of instances, subclasses, datatype/object properties) and statistical information (e.g., the cardinality of each property, min, max value for datatype properties' ranges, etc.) are provided.

Finally, the attenders can interactively modify the hierarchy specifications. Particularly, they are able to increase or decrease the level of abstraction/detail presented, by modifying modifying both the number of hierarchy levels, and number of nodes per level.

4 Related Work

A large number of works studying issues related to RDF or LOD visualization and analysis have been proposed in the literature [2,3,4,5]. Additionally, numerous tools offering RDF or Linked Open Data visualization have been developed, e.g., *Sgvizler* [6], *LODWheel* [7], *Payola* [8], *CubeViz* [9], *KC-Viz* [10], *RelFinde*[4], *Welkin*[5], *IsaViz*[6], *RDF-Gravity*[7], etc.

In the context of RDF and Linked Open Data statistics, *RDFStats* [14] calculates statistical information about RDF datasets. *LODstats* [11] is an extensible framework, offering scalable statistical analysis of Linked Open Data datasets.

Regarding the quality assessment issues, [13] studies the criteria which can be used in Linked Data quality assessment. [14] review millions of RDF documents to analyse Linked Data conformance. Finally, several frameworks for the quality assessment in the Web of Data, have been proposed *LINK-QA* [15], *Sieve* [16], *WIQA* [17]. In contrast to existing approaches, we provide hierarchical RDF data visualization enriched with data statistics. The hierarchical model solves the visualization overload issues, offering efficient, on the fly statistical computations over hierarchy levels. Finally, due to hierarchical model our tool can efficiently handle and analyse very large datasets.

5 Conclusions

In this paper we have presented rdf:SynopsViz, a framework for hierarchical charting and exploration of Linked Open Data. The hierarchical model adopted by our framework can address the problem of information overloading, offering an effective mechanism for information abstraction and summarization. Additionally, the adopted model allows the efficient statistic computations, using aggregations over the hierarchy levels.

[4] www.visualdataweb.org/relfinder.php
[5] simile.mit.edu/welkin
[6] www.w3.org/2001/11/IsaViz
[7] semweb.salzburgresearch.at/apps/rdf-gravity

Some future extensions of our tool include the application of more sophisticated filtering techniques (e.g., SPARQL-enabled browsing over the data), as well as the addition of more visual techniques and libraries.

Acknowledgement. This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS and KRIPIS - Investing in knowledge society through the European Social Fund.

References

1. Elmqvist N., Fekete J-D., “Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines.”, IEEE Trans. Vis. Comput. Graph. 16(3) 2010
2. Dadzie A., Rowe M.,“Approaches to visualising Linked Data: A survey”, Semantic Web 2(2), 2011
3. Brunetti J., Auer S., Garcia R., “The Linked Data Visualization Model”, ISWC 2012
4. Dadzie A., Rowe M., Petrelli D.“Hide the Stack: Toward Usable Linked Data”, ESWC 2011
5. Alonen M., Kauppinen T., Suominen O., Hyvonen E. “Exploring the Linked University Data With Visualization Tools”, ESWC 2013
6. Skjveland M.: “Sgvizler: A JavaScript Wrapper for Easy Visualization of SPARQL Result Sets”, ESWC 2012
7. Stuhr M., Dumitru R., Norheim D.,“LODWheel - JavaScript-based Visualization of RDF Data”, Workshop on Consuming Linked Data 2011
8. Klímek J., Helmich J., Necaský M., “Payola: Collaborative Linked Data Analysis and Visualization Framework”, ESWC 2013
9. Salas P., Mota F., Breitman K., Casanova M., Martin M., Auer S., “Publishing Statistical Data on the Web”, IEEE Semantic Computing 2012
10. Motta E., Mulholland P., Peroni S., d’Aquin M., Gomez-Perez J., Mendez V., Zablith F., “A Novel Approach to Visualizing and Navigating Ontologies”, ISWC 2011
11. Auer S., Demter J., Martin M., Lehmann J.: “LODStats - An Extensible Framework for High-Performance Dataset Analytics”, Knowledge Engineering and Knowledge Management 2012
12. Langegger A., Wöß W.: “RDFStats - An Extensible RDF Statistics Generator and Library” Workshop on Web Semantics 2009
13. Zaveri A., Rula A., Maurino A., Pietrobon R., LehmannJ., Auer S.: “Quality assessment methodologies for linked open data”, Under review, available at Semantic Web Journal site.
14. Hogan A., Umbrich J., Harth A., Cyganiak R., Polleres A., Decker S.: “An empirical survey of Linked Data conformance.”, J. Web Sem. 14, 2012
15. Guéret C., Groth P. T., Stadler C., LehmannJ. “Assessing linked data mappings using network measures”, ESWC 2012.
16. Mendes P., Mühleisen H., Bizer C.: “Sieve: linked data quality assessment and fusion”, Workshop on Linked Web Data Management 2012
17. Bizer C., Cyganiak R. “Quality-driven information filtering using the WIQA policy framework”, J. Web Sem. 7(1) 2009

RDF Resource Search and Exploration with LinkZoo

Marios Meimaris[1,2], Giorgos Alexiou[1,3] , Katerina Gkirtzou[1], George Papastefanatos[1] and Theodore Dalamagas[1]

[1]Institute for the Management of Information Systems, Reseach Center ATHENA, Athens, Greece
[2]Department of Computer Science and Biomedical Informatics, University of Thessaly, Volos, Greece
[3]Department of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece
{m.meimaris, galexiou, kateina.gkirtzou, gpapas, theo }@imis.athena-innovation.gr

Keywords: Linked Data, RDF, Collaboration, Graph Search

Abstract: The Linked Data paradigm is the most common practice for publishing, sharing and managing information in the Data Web. Linkzoo is an IT infrastructure for collaborative publishing, annotating and sharing of Data Web resources, and their publication as Linked Data. In this paper, we overview LinkZoo and its main components, and we focus on the search facilities provided to retrieve and explore RDF resources. Two search services are presented: (1) an interactive, two-step keyword search service, where live natural language query suggestions are given to the user based on the input keywords and the resource types they match within LinkZoo, and (2) a keyword search service for exploring remote SPARQL endpoints that automatically generates a set of candidate SPARQL queries, i.e., SPARQL queries that try to capture user's information needs as expressed by the keywords used. Finally, we demonstrate the search functionalities through a use case drawn from the life sciences domain.

1 INTRODUCTION

The Data Web has completely changed the way we create, interlink and consume large volumes of information. More and more corporate, 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 Data Web extents current Web infrastructure to a global data space containing and connecting data from very diverse domains.

The Linked Data paradigm is the most common practice for publishing, sharing and managing information in the Data Web, and offers a new way of data integration and interoperability. The main concept in Linked Data is that all resources published on the Web are uniquely identified by a URI, and typed links (instead of traditional Web hyperlinks) between URIs are used to semantically connect resources. Reusing existing URIs rather than creating new ones, and pointing from one dataset to another by referencing these URIs, forms the Linked Open Data cloud (Bizer et al., 2009).

Linked Data is mainly implemented with the Resource Description Framework (RDF). An RDF representation is a set of statements about resources, known as *triples*, i.e. expressions of the form *subject predicate object*. The *subject* refers to a resource to be described. Actually, the subject is a URI reference to that resource, which identifies it unambiguously. Predicates are usually terms from existing vocabularies and ontologies and are also identified by URIs. Finally, the *object* can be either a literal or a URI that refers to another RDF resource. We will refer to triples whose objects are literals as *entity-to-attribute* properties, and to triples whose objects are entities as *inter-entities* properties. A set of RDF triples can be represented by a directed labelled graph, known as the RDF data graph. However, in practice, RDF triples are stored in relational database systems, native triple/quad stores or graph DBMS (Faye et al, 2012; Bizer & Schultz, 2008). To query Linked Data, the SPARQL query language is used (Prud'Hommeaux & Seaborne, 2008).

In this paper, we briefly describe LinkZoo (Meimaris et al., 2014), a web-based platform for collaborative management, editing and sharing of Data Web resources, and we mainly focus on the search facilities. Two LinkZoo search services are presented: (1) an interactive, two-step keyword search service, where live natural language query suggestions are given to the user based on the input keywords and the resource types they match within LinkZoo, and (2) a keyword search service for exploring remote datasets, which automatically generates a set of candidate SPARQL queries that try to capture user's information need as expressed by the keywords used.

To demonstrate the services' effectiveness, we describe a real use case where the search facilities of LinkZoo are combined to effectively address user needs when working with a scientific Linked Data set. Furthermore, we perform a preliminary effectiveness study to evaluate the keyword search service with query candidates. LinkZoo is available at: `http://www.linkzoo.gr:9000` with credentials (user: *data_2015*, password: *data_2015*) for the demo account.

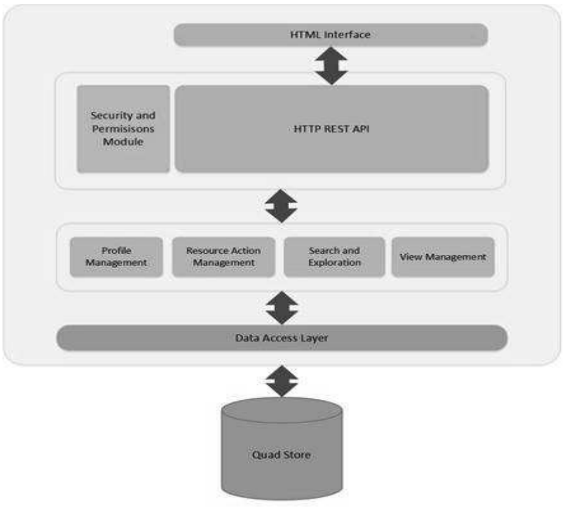


Figure 1: Linkzoo Architecture.

2 LINKZOO OVERVIEW

2.1 Architecture

The architecture of LinkZoo is shown in Figure 1. The Storage Layer is built on top of a persistent quad store, while the system's functionality is based on four basic modules: the *Profile Management module*, the *Resource Action Management module*, the *View Management module,* and the *Search module*. The *Profile Management* module is responsible for the administration of user accounts, handling actions such as user administration, account management, ascribing namespaces and named graphs to users. The *Resource Action Management* module provides processing and editing of resources such as importing, annotating and dereferencing. It is responsible for handling all actions associated with each type of resource. Also, it provides resource sharing functionality among users. The *View Management* module controls the lifecycle of views and folders; it also manages containment relationships between resources, folders and views. Views can be considered as different workspaces where the same resources can be organized in various ways. Finally, the *Search & Exploration Module* is responsible for the searching facilities implemented in LinkZoo. Specifically, it implements different search mechanisms as well as faceted browsing capabilities for private and public user graphs. A more in-depth discussion of the Search module is presented in Section 3.

2.2 Resource model

The LinkZoo Resource Representation Model captures the following aspects: (i) resource descriptive metadata, (ii) resource interlinking, and (iii) view definitions and containment relationships of resources in views and folders. Common vocabularies such as RDFS, Dublin Core Terms and FOAF are used to model non-functional metadata (e.g. labels, creators, etc.). Moreover, users can import existing ontologies or define new ones under their own schema namespace. Given that a resource can co-exist in many user accounts (e.g., in case two users happened to import the same resource), resource definitions and views in LinkZoo depend on user context. Multiple parallel versions of resource definitions are stored in their owners' named graphs. LinkZoo handles a variety of resource types, e.g., files, URLs, contacts, RDF datasets and remote SPARQL endpoints. Furthermore, folders, i.e., resource collections, are also modelled as a special resource type, and, thus, can be annotated, shared and linked accordingly. We have defined an extensible taxonomy of resource types that includes various levels of specialization for each type. This way, we allow for different handling of each resource type or sub-type.

2.3 Resource Annotations

In LinkZoo, users can annotate resources and enrich their definition with new triples. Many established ontologies and vocabularies have been imported in the tool for quick access, while new properties can also be created on demand, under each user's custom schema namespace. Annotation can be performed manually and collaboratively, as well as automatically. In the case of URL resources, external APIs are used to automatically enrich the imported URLs by parsing their content and extracting related Linked Data entities (specifically DBPedia and Freebase resources). In particular, LinkZoo utilizes the Alchemy API (Alchemy API, 2015) but other similar services can be used as well.

2.4 Sharing and Collaboration

LinkZoo resources can be collaboratively annotated and enriched with new knowledge. This is achieved by sharing resources with other users, with appropriate roles and privileges. Three levels of privileges, represented by three user roles, ensure proper sharing and usage among users. These are the *owner*, *editor* and *viewer*. Owners and editors of resources are able to share/unshare, annotate and delete them. Viewers cannot perform any kind of write-related operation that alters the state of the resource in the storage, and are thus limited to read-only actions of their shared resources. Furthermore, resources can be *private* or *public*. Shared directories pass on their sharing status to their contained items, and whenever a new resource is inserted into a folder, it automatically becomes available to the folder's shared users.

2.5 Linked Data Publication

Creating and publishing resources as Linked Data is a key feature of LinkZoo. This means that created resources are automatically assigned dereferenceable URIs, which can be used for external linking and referencing. These URIs follow a simple minting schema that takes into account the type of resource as well as a unique identifier created dynamically.

Dereferencing is performed when there are appropriate permissions, thus restricting external users with no authorization from getting access to descriptions of private resources. Unauthorized dereferencing returns a limited description. However, for a private resource, a public dereferenceable URI can be generated on demand, allowing the user to offer access to others without changing its status. If the user owns a LinkZoo account, he can choose to import the item to his account. Also, the platform offers serialized RDF exporting facilities for selected resources.

2.6 Static and Dynamic Views

The default exploring and browsing mode of LinkZoo follows the traditional folder-based approach of file systems with visual interfaces. However, LinkZoo exploits the semantics of the resources to provide multiple ways of organization. Users are able to organize their resources based on their properties and store the results as linked views. Views leverage the semantic web by offering intuitive means for organizing, searching and discovering new resources either within the platform or the entire LOD cloud. In essence, views act as workspaces and can be specialized in two sub-types, namely *static* and *dynamic*. These can be parallelized with materialized and non-materialized views in relational models respectively. Dynamic views are result sets of particular queries that are associated with the views. This way, the various annotations of resources are used as organizational factors, depending on the user's needs. For instance, the user can create a dynamic view with the query "*Find all hairpins that produce mature with name hsa*-mir-147a". This will organize into a dedicated workspace all resources that are matched by the evaluation of this query. Updating the dynamic view will result in repopulating the view based on the updated query result set.

3 RESOURCE SEARCH AND EXPLORATION

Search and exploration in LinkZoo combines keyword-based search with property-based faceted browsing. Specifically, we have implemented an "*on-the-go*" search mechanism that serves suggestions based on the taxonomy of resource types as well as the properties of resources. This type of search is applied on LinkZoo resources that have been imported to or shared with a user's account. Furthermore, we have implemented a "*search–with-query-candidates*" mechanism that can be used to query remote endpoints imported in LinkZoo. This way, LinkZoo allows exploring, importing, and annotating remote datasets. Therefore, by combining the search functionalities, users can first find relevant endpoints and then query them explicitly.

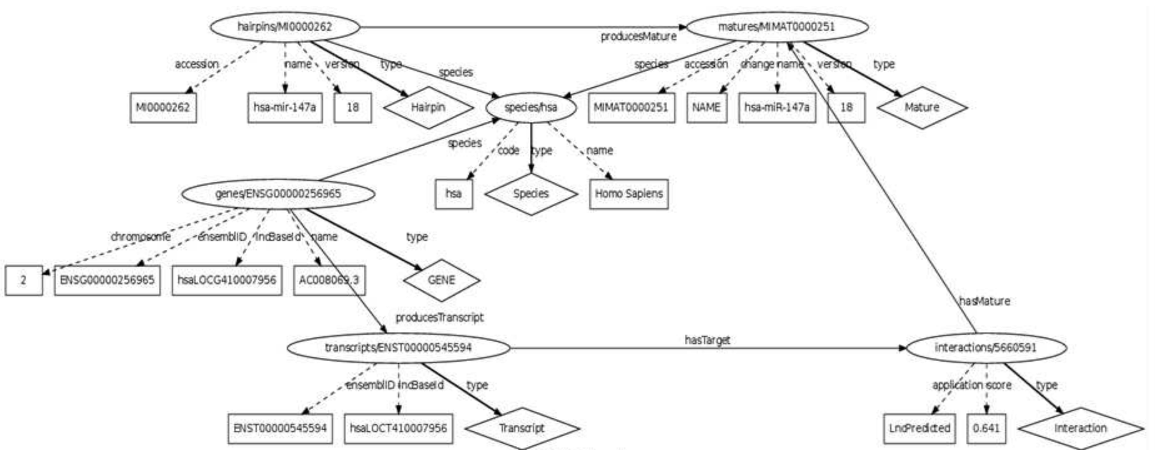


Figure 2. Example of an RDF data graph.

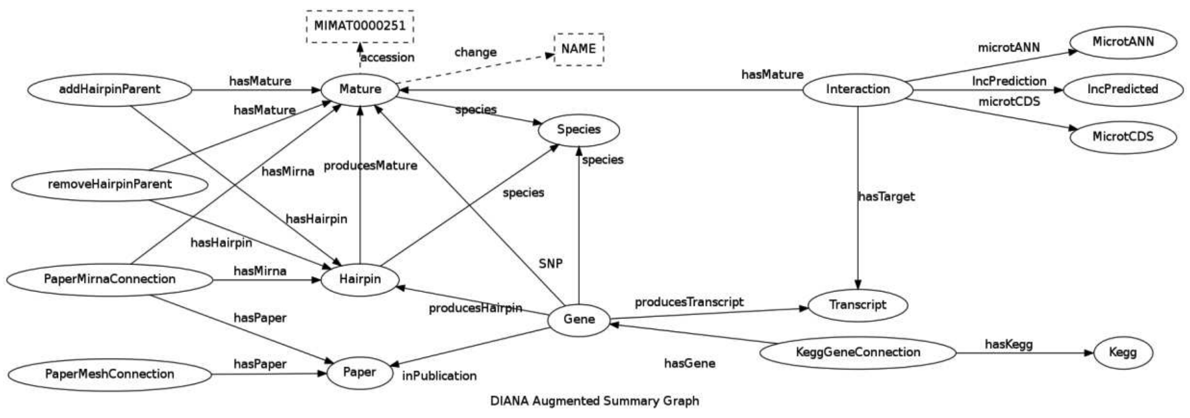


Figure 3. One of the 5 possible Augmented Summary Graphs for the keywords MIMAT0000251, name and hasTarget for the data graph shown in Figure 2.

LinkZoo also provides exploration by faceted browsing. In every folder shown to the user, the system also shows all properties from triples with the contained resources as subjects. Then, upon selection of a property, the objects in the triples of that property will be listed in the form of virtual folders for further exploration. For instance, in a folder that contains MP3 audio files, the property `mo:genre` will be selectable for faceted browsing. Then, the MP3 resources will be organized to virtual folders based on the distinct values of the `mo:genre` property. The keyword search and property filtering methods can be combined and applied in an exploratory manner.

3.1 Search with on-the-go suggestions

An interactive, two-step search process is implemented for the exploration of the user's imported and shared data within LinkZoo. Natural language query suggestions are given to the user based on the input keywords and the resource types they match. For instance, by typing "*Research*", the user will be prompted to select a suggestion from a list that contains queries of the form "*find URLs with rdfs:seeAlso dbpedia:Research*", if a similar pattern exists in the user's data. Upon selection of a suggested query, the relevant results will be shown in a virtual folder. These can then be further processed in bulk for annotation, moving, deleting and other resource-specific operations. This kind of search works incrementally, on the results previously fetched. For example, after the user selects "*find URLs with rdfs:seeAlso dbpedia:Research*" and the relevant results are fetched, further keyword exploration will suggest queries based only on the current result set and not on the whole set of user data. The above points can be summarized in the following steps: *(1)* Each keyword entered by the user is matched to objects of the triples of the user's resources. The distinct predicate-object pairs that are matched are ordered by their resource types. *(2)* The system feeds back suggestions of the form "find {resource_type} with {predicate} {object} ". For example, "*find URLs with rdfs:seeAlso dbpedia:Research". (3)* The user selects a suggestion and the system builds a query based on the resource type and the predicate object pairs found in (1). *(4)* The system feeds back the results of the query in (3) to the user. *(5)* The user goes back to (1) and enters a new keyword in order to refine the results.

3.2 Search with query candidates

A key feature of LinkZoo is a search service that assists the user to explore remote RDF data sources and to retrieve RDF entities, which, in turn, can be imported in LinkZoo. Given a set of keywords, LinkZoo returns a set of candidate SPARQL queries that try to capture user's information need as expressed by the keywords. Briefly, given a set of *n* keywords, we perform the following steps: *(1)* For each keyword k_i, we retrieve all its matches M_i on the RDF data graph. *(2)* We calculate all possible combinations $C = M_1 \times M_2 \times \cdots \times M_n = \{ c = (m_1, \cdots, m_n) | m_i \in M_i \, \forall i = 1, \cdots, n\}$ of all the matched elements M_i where $i = 1, \cdots, n$. *(3)* For each combination $c \in C$ that contains one matched element m_i per keyword k_i, we create an *augmented summary graph* G_c. *(4)* From each augmented summary graph G_c, we generate the *query pattern graph* G_c^{QP} and finally *(5)* we translate each query pattern graph G_c^{QP}. into a SPARQL query. Next, we elaborate on the details.

Let's consider that we have an RDF dataset as the one shown in Figure 2. The dataset is depicted as an RDF data graph, where oval shape vertices represent RDF entities, diamond shape vertices represent RDF classes and rectangle shape vertices represent literals. Similarly, dashed edges represent entity-to-attribute properties, while solid ones represent inter-entities properties. Let's us assume that the user has provided the keywords *MIMAT0000251, name* and *hasTarget*. The first step is to match the keywords to elements in the RDF data graph: (1) *MIMAT0000251* matches to the literal "MIMAT0000251" that is connected via the property "accession" with an RDF entity of "Mature" type, (2) *name* matches to the literal "NAME" that is connected via the property "change" met with RDF entity of type "Mature" and to the entity-to-attribute property "name" met with entities of type "Hairpin", "Mature", "Species" and "Gene", resulting in 5 possible matches, and (3) *hasTarget* matches to the inter-entities property "hasTarget" met with subject of type "Interaction" and object of type "Transcript". The second step is to calculate all possible combinations of the matched elements and for each combination c create the augmented summary graph G_c. In this example, there are 5 possible combinations. Let's examine one combination, where the *name* keyword matches to the literal "NAME".

Augmented summary graph. The augmented summary graph G_c is a combination of an aggregated representation of the RDF data graph G, enriched with graph elements for each matched element $m_i, i = 1, \cdots, n$. More specifically, all entities from the RDF data graph G that have the same type of RDF class are represented by a vertex labelled with the name of the RDF class. Similarly, all inter-entities properties of the same type are represented by a directed edge between the aggregated vertex representation of the subjects and the aggregated vertex representation of the objects. The edge is also labelled with the property's name. Note that entity-to-attribute properties as well as literal values are omitted from the summary representation. Overall, the augmented graph is actually an abstraction of the RDF data graph G.

The augmented summary graph G_c contains also graph elements for each element m_i from the set of matched elements c. More specifically, if the matched element m_i is a literal value, then the graph is extended by a directed edge and a vertex. The edge represents the entity-to-attribute property that the matched element is met with in the RDF data graph, while the vertex is the matched element m_i itself. Note that the edge is attached from the aggregated vertex representation of the subject to the newly inserted vertex. Similarly, if the matched element m_i is an entity-to-attribute property, then the graph is extended by a directed edge and a vertex. The edge represents the entity-to-attribute property, i.e. the matched element m_i, and it is attached from the aggregated vertex representation of the subject to the newly inserted vertex. The difference from the previous case is that the latter vertex represents the unknown literal of the property. Note that if the same entity-to-attribute property is met with multiple RDF entities of different RDF types in the RDF data graph that would lead to different sets c. An example of the Augmented Summary Graph for the combination under investigation is shown in Figure 3.

Query pattern graph. In order to extract the query pattern graph G_c^{QP} from the augmented summary graph G_c, we calculate the shortest paths between every pair of matched elements and we combine all of them into one connected subgraph. 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_c.

For the Augmented Summary Graph of Figure 3, since we have three keywords, we need to calculate three shortest paths. We then combine the shortest paths into a single connected component, resulting to the query pattern graph shown in Figure 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 shortest paths.

Candidate SPARQL generation. The final step of the mapping process is to translate the query pattern graph G_c^{QP} into a SPARQL query. Note that the vertices of the query pattern graph G_c^{QP} are either known or unknown literal values and aggregated representations. We need to connect the latter type of vertices and the vertices of unknown literal values with variables in order to form the SPARQL triple patterns. Note that labels of the vertices can be used as constants in the triple patterns, while the labels of the edges as predicates. To produce conjunctive SPARQL queries, given the above observations for every vertex $v \in G_c^{QP}$, we perform the following:

- if v is a literal, do nothing
- if v is an unknown literal, then connect the vertex into a new variable $var(v)$.
- If v is a aggregated representation for entities of RDF type *class* with label $label(v) = class$, then the vertex is connected into a new variable $var(v)$ and produce the following SPARQL triple $var(v)$ `rdf:type` $label(v)$.

Similarly, for every edge $e \in G_c^{QP}$:

- If e represents an inter-entities property between a vertex $subject$ and a vertex $object$, then we produce the triple pattern $var(subject)$ $label(e)$ $var(object)$.
- If e represents an entity-to-attribute property between a vertex $subject$ and a vertex $object$ that is a literal, then we produce the triple pattern $var(subject)$ $label(e)$ $label(object)$.
- If e represents an entity-to-attribute property between a vertex $subject$ and a vertex $object$ that is an unknown literal, then we produce the triple pattern $var(subject)$ $label(e)$ $var(object)$.

In our example, the pattern graph of Figure 4 is mapped to the following SPARQL query.

```
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 "MIMAT0000251".
?M diana:change "NAME".}
```

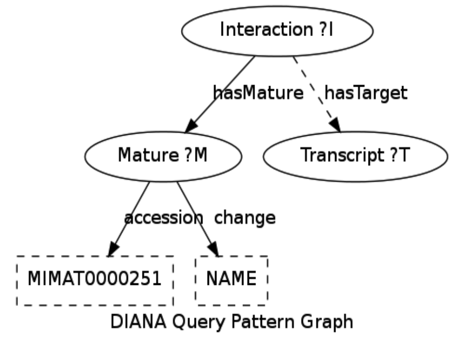


Figure 4. The Query Pattern Graph extracted from the Augmented Summary Graph of Figure 3. In dashed style, we depict the matched elements.

4 DEMONSTRATION

In this section we demonstrate the search capabilities of our tool. We employ a use case taken from the DIANA linked dataset. The dataset contains aggregated information from well-known biology databases, including ENSEMBL, miRBase and KEGG pathway, of the microRNA world published in RDF, available at the endpoint `http://leonardo.imis.athena-innovation.gr:8891/diana/sparql`.

Let us consider the following scenario: a user is engaged in a bioinformatics research project which concerns control mechanisms for cancer studies, and more specifically it focuses on the regulatory microRNA molecules. To this end, the user has gathered resources and data from a variety of sources, such as publications from PubMed, and data from the Gene Expression Atlas, the Experimental Factor Ontology and DIANA. Some of these resources have been imported by the user himself, while others have been shared to him by his collaborators. Publications are modelled either with the *file* or *URL* type and are annotated with metadata provided by the user and collaborators as well as external enrichment services. On the other hand, the imported datasets are modelled as resources of the type *DataCollection*, which allows the exploration of a remote RDF dataset via our *search-with-query-candidates* mechanism. Similarly to other resource types, *DataCollection* resources are annotated with descriptive metadata.

We consider that the user has either limited knowledge of the RDF vocabulary used to describe the datasets, or limited experience with SPARQL. To overcome this problem, LinkZoo offers the capability of keyword search for identifying and exploring RDF datasets. The user can identify potential datasets that fit his criteria, based on metadata annotations. In this case, he is looking for datasets that contain microRNA data, and to that end he uses the *search-on-the-go* utility to eventually identify the DIANA dataset.

After identifying DIANA as the dataset to work with, the user is interested in collecting information about zebrafish miRNAs, in order to evaluate a potential correlation with human cancer cell metastasis. To achieve this, he types the words "*zebrafish hairpin*" into the keyword search box and as a result he gets two possible SPARQL queries. Both generated queries will search for publications that are annotated with the mesh term "*zebrafish*" and are related with miRNAs of hairpin type. In the first query, the "*hairpin*" keyword matches to the RDF class `diana:Hairpin` and imposes a direct constraint that the property `diana:hasMirna` of the RDF class `diana:PaperMirnaConnection` will retrieve only Hairpin entities, while in the second one the "*hairpin*" keyword matches to the literal value of the property `diana:mirnaType` of the RDF class `diana:PaperMirnaConnection`, imposing an indirect constraint to the property `diana:hasMirna`. Moreover, the first query will also retrieve the RDF entities of the connected Hairpins, while the second will not. The data he retrieved from the keyword search request, could provide useful information that would allow the user to further explore the dataset. Also, the user can retrieve results by selecting one of the generated queries, and incorporate them as new resources in his LinkZoo account, in order to annotate them and share them with his collaborators.

4.1 Preliminary Evaluation

In order to evaluate the search with query candidates, we perform an effectiveness study. We have asked our biologists collaborators to provide keyword queries along with a description in natural language of the required information. We have aggregated 5 queries for the DIANA dataset. An example query is "*"Alzheimer's disease" mature"* and the corresponding description is *"Retrieve all mature miRNAs that are related with Alzheimer's disease"*. To evaluate the effectiveness of our generated queries we order them in reverse order given the number of triple patterns they contain and we calculate the Reciprocal Rank metric defined as RR = 1/r where r is the rank of the correct query. Given our problem definition, a query is correct if it matches the information needs as explained in the provided natural language description. Figure 5 shows the Reciprocal Rank we have calculated for the 5 queries for the DIANA dataset. In the 4 out of 5 queries, we got an RR of 1 meaning that we were able to get the information required by the users.

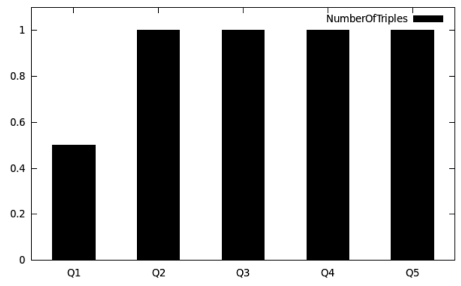


Figure 5. Reciprocal Rank for the DIANA dataset.

5 RELATED WORK

Collaborative editing and annotating has been explored and addressed thoroughly on the schema level. Tools available for collaborative ontology editing are presented in (Auer et al., 2006; Farquhar et al., 1997; Tudorache et al., 2013). However, these require expertise on the schema level. Regarding the management of heterogeneous resources, Personal Information Management (PIM) systems and tools have been implemented, employing common representation semantics as an abstraction layer (Bernardi et al., 2011; Franz et al., 2007; Sauermann et al., 2006). However, these address the management of resources in non-collaborative communities and are thus limited to individual usage.

On the other hand, the keyword search problem over structured data, tree structured (Cohen et al., 2003; Kimelfeld & Sagiv, 2006) or graph structured (He et al., 2007; Bhalotia et al., 2002), is a problem that has widely been explored. Basic steps in those works involve 1) mapping the keyword elements to data elements 2) searching for substructures on the data that connect the keyword elements and 3) return as output the substructures given a scoring function.

(Tran et al., 2009) proposed a different solution to the keyword search problem, where instead of computing for the answers directly, it computes structured queries allowing the user to choose the appropriate one. LinkZoo's approach on keyword search with query candidates follows (Tran et al., 2009) approach to generate SPARQL queries, but uses a different exploratory method. In particular, we create multiple augmented graphs one per keywords combination and use the notion of shortest paths to create a query pattern graph.

6 CONCLUSIONS

In this paper, we have presented LinkZoo, an IT infrastructure for collaborative management of heterogeneous resources on the Web. LinkZoo provides an environment for modelling and publishing data such as files, websites, datasets and people as RDF, and allows for their coexistence in shared contexts. Furthermore, we have presented the various types of search capabilities implemented in the platform. These span from trivial text searching within a user's data to more elaborate data-guided exploration and searching over imported RDF data collections. Finally, we have demonstrated the usability of the search functions through a use case drawn from the life sciences domain.

Currently, keyword search expects exact matches of terms. In the future, we will extend this functionality to automatically suggest terms from the RDF data graph. Another direction will be to extend the matching procedure by enabling also ontology matching (Euzenat & Shvaiko, 2013). Furthermore, to assist user understanding of the produced candidate SPARQL queries, we intend to also show natural language descriptions of the generated candidates. Finally, we also plan to perform an extensive evaluation of our search services, in term of completeness of the results, time and memory requirements for indices creation, performance.

ACKNOWLEDGEMENTS

This study has been partially supported by LODGOV project, Research Programme ARISTEIA (EXCELLENCE), General Secretariat for Research and Technology, Ministry of Education, Greece and the European Regional Development Fund, and the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund.

REFERENCES

AlchemyAPI | Powering the New AI Economy. Available from:<http://www.alchemyapi.com/>.[2015].

Auer, S., et al. 2006, 'OntoWiki–a tool for social, semantic collaboration', *Proceedings ISWC,* Springer Berlin Heidelberg, pp. 736-749.

Bernardi, A., et al., 2011, 'The NEPOMUK semantic desktop', *Proceedings CSKM,* Springer Berlin Heidelberg, pp. 255-273.

Bhalotia, G., et al., 2002, 'Keyword searching and browsing in databases using BANKS', *Proceedings ICDE,* pp. 431-440.

Bizer, C., Heath, T., & Berners-Lee, T., 2009, 'Linked data-the story so far'

Bizer, C., & Schultz, A., 2008, 'Benchmarking the performance of storage systems that expose SPARQL endpoints', *Proceedings WWW.*

Cohen, S., et al., 2003, 'XSEarch: A semantic search engine for XML', Proceedings *VLDB*, pp. 45-56.

Erling, O., 2012, 'Virtuoso, a Hybrid RDBMS/Graph Column Store', *IEEE Data Eng. Bull.*, pp. 3-8.

Euzenat, J., & Shvaiko, P., 2013, *Ontology matching* (2nd edition), Heidelberg: Springer-Verlag.

Farquhar, A., Fikes, R., & Rice, J., 1997, 'The ontolingua server: A tool for collaborative ontology construction', *International journal of human-computer studies*, vol. *46, no.* 6, pp. 707-727.

Faye, D. C., Cure, O., & Blin, G., 2012, 'A survey of RDF storage approaches', *ARIMA Journal*, vol. *15*, pp. 11-35.

Franz, T., Staab, S., & Arndt, R. , 2007, 'The X-COSIM integration framework for a seamless semantic desktop', *Proceedings K-CAP,* pp. 143-150.

He, H., et al., 2007, 'BLINKS: ranked keyword searches on graphs', *Proceedings SIGMOD,* pp. 305-316.

Kimelfeld, B., & Sagiv, Y., 2006, 'Finding and approximating top-k answers in keyword proximity search', *Proceedings SIGMOD-SIGACT-SIGART* pp. 173-182.

Meimaris, M., Alexiou, G., & Papastefanatos, G., 2014, 'LinkZoo: A linked data platform for collaborative management of heterogeneous resources', *Proceedings ESWC,* pp. 407-412.

Prud'Hommeaux, E., & Seaborne, A., 2008, 'SPARQL query language for RDF', *W3C recommendation*, *15*.

Sauermann, L., et al., 2006, 'Semantic desktop 2.0: The gnowsis experience', *Proceedings ISWC,* pp. 887-900.

Tran, T., et al., 2009, 'Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data', *Proceedings ICDE,* pp. 405-416.

Tudorache, T., et al., 2013, 'WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web', *Semantic web*, vol. *4, no.* 1, pp. 89-99.

RDivF: Diversifying Keyword Search on RDF Graphs

Nikos Bikakis[1,2], Giorgos Giannopoulos[1,2⋆], John Liagouris[1,2⋆], Dimitrios Skoutas[2], Theodore Dalamagas[2], and Timos Sellis[3]

[1]National Technical University of Athens, Greece
[2]"Athena" Research Center, Greece [3]RMIT University, Australia

Abstract. In this paper, we outline our ongoing work on diversifying keyword search results on RDF data. Given a keyword query over an RDF graph, we define the problem of diversifying the search results and we present diversification criteria that take into consideration both the content and the structure of the results, as well as the underlying RDF/S–OWL schema.

Keywords: Linked Data, Semantic Web, Web of Data, Structured Data.

1 Introduction

As a growing number of organizations and companies (e.g., *Europeana*, *DBpedia*, *data.gov*, *GeoNames*) adopt the *Linked Data* practices and publish their data in RDF format, going beyond simple SPARQL endpoints, to provide more advanced, effective and efficient search services over RDF data, has become a major research challenge. Especially, since users prefer searching with plain keywords, instead of using structured query languages such as SPARQL, there has been an increasing interest on keyword search mechanisms over RDF data [1,2,3].

Most of the proposed works return the *most relevant* RDF results, in the form of *graphs* or *trees*. Relevance, in this case, is typically defined in terms of (a) *content similarity* between the elements comprising a result and the query terms and (b) *result compactness*, which means that smaller trees or graphs are preferred. The drawback is that this leads to result sets that are often characterized by a high degree of redundancy. Moreover, significant information is often lost, since graph paths that connect two entities and might denote a significant relation between them are omitted to satisfy the compactness requirement. Moreover, most approaches do not consider the rich *structure* and *semantics* provided by the RDF data model. For instance, an effective RDF keyword search method should treat RDF properties (edges) as first-class citizens, since properties may provide significant information about the *relations* between the entities being searched.

⋆ This research has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund.

As an example, consider a user searching for "*Scarlett Johansson, Woody Allen*" over the DBpedia dataset. An effective approach should, at least initially, consider *all* the possible ways these two entities are related. Since there exist various roles and relations between these two entities, e.g., Woody Allen may appear as either a director or an actor, this leads to a large and complex result set, containing several overlapping or similar results. The plethora of different relation combinations requires a mechanism that reduces information redundancy, allowing the system to return to the user a more concise and also more meaningful and informative result set. This can be achieved by introducing a diversification step into the retrieval and ranking process. Ideally, the system should return to the user results that cover different aspects of the existing connections between these entities, e.g., a movie where they played together, a movie directed by Woody Allen where Scarlett Johansson appears, an award they shared, etc.

Although the diversification problem has been extensively studied for document search [7,8,9], the structured nature of RDF search results requires different criteria and methods. Most of the approaches regarding keyword search on graphs [1,2,5] limit their results to trees (particularly, variations of Steiner trees); only few allow subgraphs as query answers [3,4]. Among them, [3] is the most relevant to our work; however, it does not address the diversification problem and it also does not consider the schema of the data. A different perspective is followed in [6], where a keyword query is first interpreted as a set of possible structured queries, and then the most diverse of these queries are selected and evaluated.

In this paper, we introduce a diversification framework for keyword search on RDF graphs. The main challenges arise from the fact that the structure of the results, including additional information from the underlying schema, needs to be taken into account. This is in contrast to the case of diversifying unstructured data, where the factor of content (dis)similarity is sufficient. In our framework, called RDivF (RDF + Diversity), which we are currently developing, we exploit several aspects of the RDF data model (e.g., resource content, RDF graph structure, schema information) to answer keyword queries with a set of diverse results. To the best of our knowledge, this is the first work addressing the issue of result diversification in keyword search on RDF data.

2 Diversifying RDF Keyword Search

Assume an *RDF graph* $G(V, E)$, where V is the set of *vertices* and E the set of *edges*. Optionally, G may be associated with an RDF schema, defining a hierarchy among classes and properties. Let $q = \{\{t_1, t_2, \ldots, t_m\}, k, \rho\}$ be a *keyword query* comprising a set of m *terms* (i.e., keywords), a parameter k specifying the *maximum number of results* to be returned, and a parameter ρ that is used to restrict the *maximum path length* between keyword nodes (i.e., vertices), as will be explained later. Assume also a function $\mathcal{M}\colon t \to V_t$ that maps a keyword t to a set of graph nodes $V_t \subseteq V$.

Definition 1. (Direct Keyword Path). Assume two nodes $u, v \in V$ that match two terms t, s of a query q, i.e., $u \in V_t$ and $v \in V_s$. Let P be a path between u and v. P is called a *direct keyword path* if it does not contain any other node that matches any keyword of the query q.

Definition 2. (Query Result). Assume an RDF graph G and a query q. A subgraph G_q of G is a *query result* of q over G, if: (a) for each keyword t in q, there exists exactly one node v in G_q such that $v \in V_t$ (these are called *keyword nodes*), (b) for each pair of keyword nodes u, v in G_q, there exists a path between them with length at most ρ, (c) for each pair of keyword nodes u, v in G_q, there exists at most one direct keyword path between them, and (d) each non-keyword node lies on a path connecting keyword nodes.

The above definitions leed to query results that contain pair-wise connections among all the terms in the query. That is, in our framework, we are interested in results that can be graphs and not only spanning trees, which is the typical case in previous approaches. This is based on the intuition that we want to emphasize on the completeness of relationships between query terms rather than on the criterion of minimality. Note that the aspect of minimality is still taken into consideration in our definition by means of the conditions (c) and (d) above.

Now, assume a function $r\colon (G_q, q) \to [0, 1]$ that measures the *relevance* between the query q and a result G_q, and a function $d\colon (G_q, G'_q) \to [0, 1]$ that measures the *dissimilarity* between two query results G_q and G'_q. Let also $f_{r,d}$ be a monotone objective function that combines these two criteria and assigns a score to a result set $\mathcal{R}$ w.r.t. the query q, measuring how relevant the included results are to the query and how dissimilar they are to each other. We assume that $|\mathcal{R}| > k$. Then, the goal of the diversification task is to select a subset of k results so that this objective function is maximized. Formally, this can be defined as follows.

Definition 3. (Diversified Result Set). Assume an RDF graph G, a query q, and the functions r, d, and $f_{r,d}$ as described above. Let $\mathcal{R}$ denote the result set of q over G. The *diversified result set* $\mathcal{R}_k$ is a subset of the results $\mathcal{R}$ with size k that maximizes $f_{r,d}$, i.e., $\mathcal{R}_k = \underset{\mathcal{R}' \subseteq \mathcal{R}, |\mathcal{R}'| = k}{\operatorname{argmax}} f_{r,d}(\mathcal{R}')$.

Following this approach, in order to select a diversified result set for keyword queries over RDF graphs, one needs to determine appropriate functions for r, d, and $f_{r,d}$. Regarding the latter, [8] presents several objective function and studies their characteristics. The same functions can also be used in our case, since this aspect is independent from the nature of the underlying data. Therefore, we focus next on specifying the relevance and dissimilarity functions, r and d, in our setting.

3 Diversification Criteria

The main challenge for diversifying the results of keyword queries over RDF graphs, is how to take into consideration the semantics and the structured nature of RDF when defining the relevance of the results to the query and the dissimilarity among results. In this section, we outline a set of criteria for this purpose, which can be used for specifying the functions r and d, as described above.

The relevance of a result to the query takes into consideration two main factors. The first factor refers to text-based matching between the nodes in the result graph and the keywords in the query. This aspect is essentially covered by the function $\mathcal{M}$ that maps query terms to graph nodes. This function can be modified to return, for each graph node, a degree of match $m \in [0, 1]$ between this node and a corresponding query keyword. In addition, a threshold τ can be specified in the query, so that only nodes with $m \geq \tau$ are returned. The second factor refers to the fact that results should be concise and coherent. One step to ensure this is the minimality criterion included in Definition 2. Furthermore, we need to consider structural and semantic homogeneity of the result, so that the results can be more meaningful to the user. This is an intra-result measure, capturing the homogeneity among the nodes, edges and paths in the result graph. For example, this would assign a higher score to a path where all the edges are labelled with the same property. Moreover, RDF schema information can be taken into account, i.e., scoring based on class or property hierarchy and least common ancestors.

The dissimilarity among results can be defined by comparing paths between corresponding pairs of keyword nodes. This takes into account both structural properties, e.g., path lengths or common subpaths, and semantic information, i.e., classes and properties corresponded to the nodes and edges along the path. The main objective here is, for each result, to obtain paths that are similar to other paths in the result, but dissimilar to paths in other results. This objective is not restrained to textual similarity only, but takes also into account the semantic similarity of classes and properties inferred by the schema.

Acknowledgement. This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Thales. Investing in knowledge society through the European Social Fund"

References

1. Tran 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.
2. Zhou Q., Wang C., Xiong M., Wang H., Yu Y.: SPARK: Adapting Keyword Query to Semantic Search. In ISWC 2007.
3. Elbassuoni S., Blanco R.: Keyword Search over RDF Graphs. In CIKM 2011.
4. Li G., Ooi B-C, Feng J., Feng J., et.al: EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured data. In SIGMOD 2008.
5. He H., Wang H., Yang J., Yu P.: BLINKS: Ranked Keyword Searches on Graphs. In SIGMOD 2007.
6. Demidova E., Fankhauser P., Zhou X., Nejdl W.: DivQ: Diversification for Keyword Search over Structured Databases. In SIGIR 2010.
7. Drosou M., Pitoura E.: Search Result Diversification. In SIGMOD Rec., 39(1), 2010.
8. Gollapudi S., Sharma A.: An Axiomatic Approach for Result Diversification. In WWW 2009.
9. Stefanidis K., Drosou M., Pitoura E.: PerK: Personalized Keyword Search in Relational Databases through Preferences. In EDBT 2010