DOGMA: A Disk-Oriented Graph Matching Algorithm for RDF Databases

Matthias Bröcheler[1], Andrea Pugliese[2], and V.S. Subrahmanian[1]

[1] University of Maryland, USA
[2] Università della Calabria, Italy

Abstract. RDF is an increasingly important paradigm for the representation of information on the Web. As RDF databases increase in size to approach tens of millions of triples, and as sophisticated graph matching queries expressible in languages like SPARQL become increasingly important, scalability becomes an issue. To date, there is no graph-based indexing method for RDF data where the index was designed in a way that makes it disk-resident. There is therefore a growing need for indexes that can operate efficiently when the index itself resides on disk. In this paper, we first propose the DOGMA index for fast subgraph matching on disk and then develop a basic algorithm to answer queries over this index. This algorithm is then significantly sped up via an optimized algorithm that uses efficient (but correct) pruning strategies when combined with two different extensions of the index. We have implemented a preliminary system and tested it against four existing RDF database systems developed by others. Our experiments show that our algorithm performs very well compared to these systems, with orders of magnitude improvements for complex graph queries.

1 Introduction

RDF is becoming an increasingly important paradigm for Web knowledge representation. As more and more RDF database systems come "online" and as RDF gets increasing emphasis from both established companies like HP and Oracle, as well as from a slew of startups, the need to store and efficiently query massive RDF datasets is becoming increasingly important. Moreover, large parts of query languages like SPARQL increasingly require that queries (which may be viewed as graphs) be matched against databases (which may also be viewed as graphs) – the set of all possible "matches" is returned as the answer.

For example, the GovTrack database [1] tracks events in the US Congress and stores the data in RDF. RDF triple stores primarily store triples (s, p, v) where s is a subject, p is a property, and v is a value. Fig. 1(a) shows a small portion of the GovTrack dataset (we have changed the names of individuals identified in that dataset). The reader can readily see that the RDF data forms a graph where the nodes correspond to subjects and values, and the edges linking them are labeled with a property. For instance, in Fig. 1(a), we see that Jeff Ryster sponsored Bill B0045 whose subject is Health Care. This corresponds to two triples (Jeff Ryster, sponsor, Bill B0045) and (Bill B0045, subject, Health Care). A user who is using such a database might wish to ask queries

A. Bernstein et al. (Eds.): ISWC 2009, LNCS 5823, pp. 97–113, 2009.

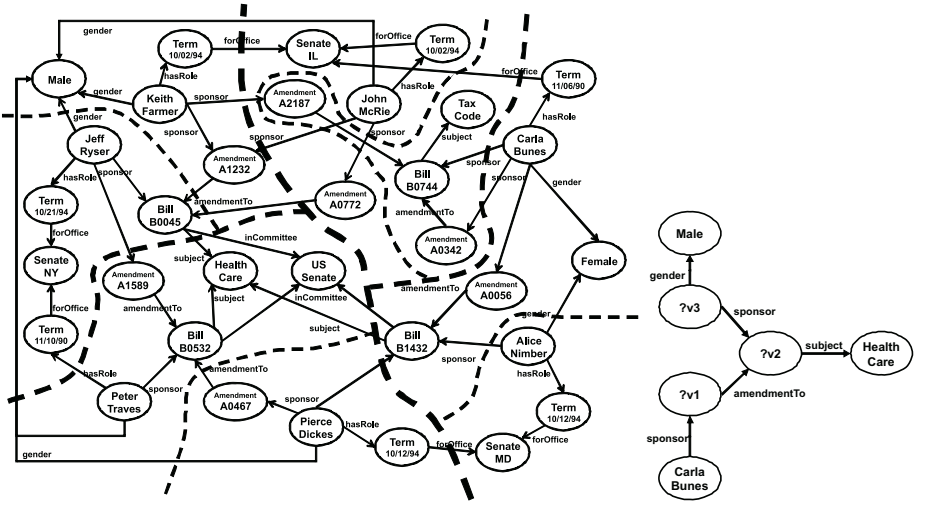

Fig. 1. Example RDF graph (a) and query (b)

such as that shown in Fig. 1(b). This query asks for all amendments ($?v1$) sponsored by Carla Bunes to bill ($?v2$) on the subject of health care that were originally sponsored by a male person ($?v3$). The reader can readily see that when answering this query, we want to find *all* matches for this query graph in the original database. The reader who tries to answer this very simple query against this very tiny database will see that it takes time to do so, even for a human being!

In this paper, we propose a *graph-based* index for RDF databases called DOGMA, that employs concepts from graph theory to efficiently answer queries such as that shown above. DOGMA is tuned for scalability in several ways. First, the index itself can be stored on disk. This is very important. From experiences in relational database indexing, it is clear that when the data is large enough to require disk space, the index will be quite large and needs to be disk resident as well. DOGMA, defined in Section 3, is the first graph-based index for RDF that we are aware of that is specifically designed to reside on disk. We define the DOGMA data structure and develop an algorithm to take an existing RDF database and create the DOGMA index for it. In Section 4, we develop algorithms to answer graph matching queries expressible in SPARQL [2] (we emphasize that we do not claim DOGMA supports all SPARQL queries yet). Our first algorithm, called DOGMA_basic, uses the index in a simple manner. Subsequently, we provide the improved algorithm DOGMA_adv and two extensions of the index called DOGMA_ipd and DOGMA_epd, that use sophisticated pruning methods to make the search more efficient without compromising correctness. Third, in Section 5, we show the results of an experimental assessment of our techniques against four competing RDF database systems (JenaTDB, Jena2, Sesame2, and OWLIM). We show that DOGMA performs very well compared to these systems.

2 Preliminaries

In this section, we briefly explain our notation. We assume the existence of a set $\mathcal{S}$ whose elements are called *subjects*, a set $\mathcal{P}$ whose elements are called *properties* and a set $\mathcal{V}$ whose elements are called *values*. Throughout this paper, we assume that $\mathcal{S}, \mathcal{P}, \mathcal{V}$ are all arbitrary, but fixed sets. If $s \in \mathcal{S}, p \in \mathcal{P}$ and $v \in \mathcal{V}$, then (s, p, v) is called an *RDF triple*. Note that $\mathcal{V}$ and $\mathcal{S}$ are not required to be disjoint. An *RDF database* is a finite set of RDF triples. For example, as mentioned earlier, (Jeff Ryster, sponsor, Bill B0045) and (Bill B0045, subject, Health Care) are RDF triples. Every RDF database $\mathcal{R}$ has an associated *RDF graph* $G_{\mathcal{R}} = (V_{\mathcal{R}}, E_{\mathcal{R}}, \lambda_{\mathcal{R}})$ where $V_{\mathcal{R}} = \mathcal{S} \cup \mathcal{V}$, $E_{\mathcal{R}} \subseteq \mathcal{S} \times \mathcal{V}$, and $\lambda_{\mathcal{R}} : E_{\mathcal{R}} \rightarrow \mathcal{P}$ is a mapping such that for all $(s, p, v) \in \mathcal{R}$, $\lambda_{\mathcal{R}}(s, v) = p$. [1] As there is a one-one correspondence between RDF graphs and RDF databases, we will often use the terms synonymously.

In this paper, we only focus on graph matching queries. In order to define such queries, we assume the existence of some set VAR of *variable symbols*. In this paper, all variable symbols will start with a "?". A *graph query* is any graph $Q = (V_Q, E_Q, \lambda_Q)$ where $V_Q \subseteq \mathsf{VAR} \cup \mathcal{S} \cup \mathcal{V}$, $E_Q \subseteq V_Q \times V_Q$, and $\lambda_Q : E_Q \rightarrow \mathcal{P}$ is a mapping. Suppose Q is a query. A *substitution* for query Q is a mapping $V_Q \cap \mathsf{VAR} \rightarrow \mathcal{S} \cup \mathcal{V}$. In other words, a substitution maps all variable vertices in query Q to either a subject or a value. For instance, in Fig. 1, the mapping θ which assigns B0744 to $?v1$, B0744 to $?v2$ and Jeff Ryster to $?v3$ is a substitution. If θ is a substitution for query Q, then $Q\theta$ denotes the replacement of all variables $?v$ in V_Q by $\theta(?v)$. In other words, the graph structure of $Q\theta$ is exactly like that of Q except that nodes labeled with $?v$ are replaced by $\theta(?v)$. A substitution θ is an *answer* for query Q w.r.t. database $\mathcal{R}$ iff $Q\theta$ is a subgraph of $G_{\mathcal{R}}$. The answer set for query Q w.r.t. an RDF database $\mathcal{R}$ is the set $\{\theta \mid Q\theta$ is a subgraph of $G_{\mathcal{R}}\}$.

Example 1. Consider the example query and RDF database in Fig. 1. In this case, the substitution θ such that $\theta(?v1)$ = Amendment A0056, $\theta(?v2)$ = Bill B1432, and $\theta(?v3)$ = Pierce Dickes is the only answer substitution for this query. □

3 The DOGMA Index

In this section, we develop the DOGMA index to efficiently answer graph queries in situations where the index itself must be very big (which occurs when $\mathcal{R}$ is very big). Before we define DOGMA indexes, we first define what it means to merge two graphs.

Suppose G is an RDF graph, and G_1 and G_2 are two RDF graphs such that $V_1, V_2 \subseteq V_{\mathcal{R}}$ and k is an integer such that $k \leq \max(|V_1|, |V_2|)$. Graph G_m is said to be a *k-merge of graphs G_1, G_2 w.r.t. G* iff: (*i*)$|V_m| = k$; (*ii*) there is a *surjective* (i.e. onto) mapping $\mu : V_1 \cup V_2 \rightarrow V_m$ called the *merge mapping* such that $\forall v \in V_m$, $rep(v) = \{v' \in V_1 \cup V_2 \mid \mu(v') = v\}$, and $(v_1, v_2) \in E$ iff there exist $v_1' \in rep(v_1), v_2' \in rep(v_2)$ such that $(v_1', v_2') \in E$. The basic idea tying k-merges to the DOGMA index is that we want

[1] For the sake of simplicity, we ignore many features in RDF such as reification, containers, blank nodes, etc. Moreover, we define $E_{\mathcal{R}} \subseteq \mathcal{S} \times \mathcal{V}$ for notational convenience; our implementation allows for multiple edges between vertices.

DOGMA to be a binary tree each of whose nodes occupies a disk page. Each node is labeled by a graph that "captures" its two children in some way. As each page has a fixed size, the number k limits the size of the graph so that it fits on one page. The idea is that if a node N has two children, N_1 and N_2, then the graph labeling node N should be a k-merge of the graphs labeling its children.

A DOGMA index for an RDF database $\mathcal{R}$ is a generalization of the well known binary-tree specialized to represent RDF graph data in a novel manner.

Definition 1. A DOGMA index of order k ($k \geq 2$) *is a binary tree* $\boldsymbol{D}_{\mathcal{R}}$ *with the following properties:*

1. *Each node in* $\boldsymbol{D}_{\mathcal{R}}$ *equals the size of a disk page and is labeled by a graph.*
2. $\boldsymbol{D}_{\mathcal{R}}$ *is balanced.*
3. *The labels of the set of leaf nodes of* $\boldsymbol{D}_{\mathcal{R}}$ *constitute a partition of* $G_{\mathcal{R}}$.
4. *If node* N *is the parent of nodes* N_1, N_2*, then the graph* G_N *labeling node* N *is a* k*-merge of the graphs* G_{N_1}, G_{N_2} *labeling its children.*

Note that a single RDF database can have many DOGMA indexes.

Example 2. Suppose $k = 4$. A DOGMA index for the RDF graph of Fig. 1(a) might split the graph into the 8 components indicated by dashed lines in Fig. 1(a) that become the leaf nodes of the index (Fig. 2). Consider the two left-most leaf nodes. They can be 4-merged together to form a parent node. Other leaf nodes can also be merged together (due to space constraints, the results of k-merging are not shown in the inner nodes). □

Even though many different DOGMA indexes can be constructed for the same RDF database, we want to find a DOGMA index with as few "cross" edges between subgraphs stored on different pages as possible. In other words, if node N is the parent of nodes N_1, N_2, then we would like relatively fewer edges in $\mathcal{R}$ between some node

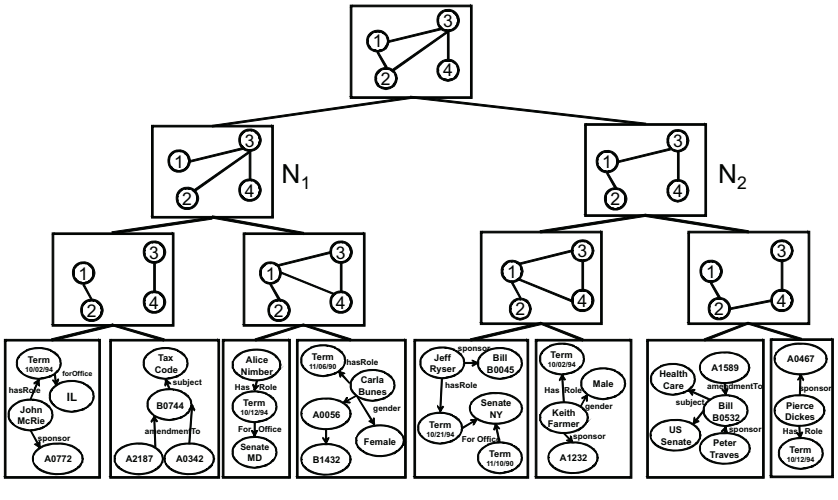

Fig. 2. A DOGMA index for the RDF database of Fig. 1(a)

in G_{N_1} and some node in G_{N_2}. The smaller this number of edges, the more "self-contained" nodes N_1, N_2 are, and the less likely that a query will require looking at both nodes N_1 and N_2. In the description of our proposed algorithms, we employ an external graph partitioning algorithm (many of which have been proposed in the literature) that, given a weighted graph, partitions its vertex set in such a way that (*i*) the total weight of all edges crossing the partition is minimized and (*ii*) the accumulated vertex weights are (approximately) equal for both partitions. In our implementation, we employ the *GGGP* graph partitioning algorithm proposed in [3].

Fig. 3 provides an algorithm to build a DOGMA index for an RDF graph $G_{\mathcal{R}}$. The BuildDOGMAIndex algorithm starts with the input RDF graph, which is set to G_0. It assigns an arbitrary weight of 1 to each vertex and each edge in G_0. It iteratively coarsens G_0 into a graph G_1 that has about half the vertices in G_0, then coarsens G_1 into a graph G_2 that has about half the vertices as G_1, and so forth until it reaches a G_j that has k vertices or less.

```
   Algorithm BuildDOGMAIndex
   Input: RDF graph G_R, page size k
          (level L, colors C)
   Output: DOGMA index D_R
 1 G_0 ← G_R
 2 for all v ∈ V_R
 3    weight(v) ← 1
 4 for all e ∈ E_R
 5    weight(e) ← 1
 6 i ← 0
 7 while |G_i| > k
 8    i ← i + 1
 9    G_i, μ_i ← CoarsenGraph(G_{i-1})
10 root(D_R) ← a new "empty" node R
11 BuildTree(R, i, G_i)
12 ColorRegions(L, D_R, C) /* Only required for the DOGMA_epd index discussed later */
13 return D_R
```

```
   Algorithm CoarsenGraph
   Input: RDF graph G_R
   Output: Coarsened graph G'_R, merge mapping μ
 1 G'_R ← G_R
 2 μ ← (V_R → V'_R) /* identity map */
 3 while 2 × |V'_R| > |V_R|
 4    v ← uniformly random chosen vertex from V'_R
 5    N_v ← {u | (u, v) ∈ E'_R}
 6    m ← x ∈ N_v s.t. x ≥ y ∀y ∈ N_v
 7    weight(m) ← weight(m) + weight(v)
 8    for all (v, u) ∈ E'_R
 9       if (m, u) ∈ E_R
10          weight((m, u)) ← weight((m, u))
11                +weight((v, u))
12       else
13          E'_R ← E'_R ∪ {(m, u)}
14          weight((m, u)) ← weight((v, u))
15    V'_R ← V'_R \ {v}
16    μ(μ^{-1}(v)) ← m
17    E'_R ← E'_R \ {(v, u) ∈ E'_R}
18 return G'_R, μ
```

```
   Algorithm BuildTree
   Input: Binary tree node N, level i,
          subgraph S at level i
   Output: Graph merge hierarchy {G_j}_{j≥0}
          and merge mappings {μ_j}_{j>0}
 1 label(N) ← S
 2 if |S| > k
 3    S_1, S_2 ← GraphPartition(S)
 4    L ← leftChild(N)
 5    R ← rightChild(N)
 6    S_L ← induced subgraph in G_{i-1}
 7          by vertex set {v | μ_i(v) ∈ V_{S_1}}
 8    S_R ← induced subgraph in G_{i-1}
 9          by vertex set {v | μ_i(v) ∈ V_{S_2}}
10    BuildTree(L, i − 1, S_L)
11    BuildTree(r, i − 1, S_R)
12 P_N ← {v | μ_i(μ_{i-1}(... μ_1(v))) ∈ V_S}
13 for all v ∈ P_N /* Only for DOGMA_ipd */
14    ipd(v, N) ← min_{u∈V_0\P_N} d_{G_0}(u, v)
```

Fig. 3. BuildDOGMAIndex, CoarsenGraph, and BuildTree algorithms

The coarsening is done by invoking a CoarsenGraph algorithm that randomly chooses a vertex v in the input graph, then it finds the immediate neighbors N_v of v, and then finds those nodes in N_v that are best according to a total ordering $\succeq$. There are many ways to define $\succeq$; we experimented with different orderings and chose to order by increasing edge weight, then decreasing vertex weight. The CoarsenGraph algorithm appropriately updates node and edge weights and then selects a maximally weighted node, denoted m, to focus the coarsening on. The coarsening associated with the node v merges neighbors of the node m and m itself into one node, updates weights, and removes v. Edges from m to its neighbors are removed. This process is repeated till we obtain a graph which has half as many vertices (or less) than the graph being coarsened. The result of CoarsenGraph is a k-merge where we have merged adjacent vertices. The BuildDOGMAIndex algorithm then uses the sequence $G_0, G_1, \ldots, G_j$ denoting these coarsened graphs to build the DOGMA index using the BuildTree subroutine. Note that Line 12 in the BuildDOGMAIndex algorithm (where L denotes the level at which to color the subgraphs and C is a list of unique colors) is only needed for the DOGMA_epd index introduced later, as well as lines 12–14 in BuildTree are for the DOGMA_ipd index. They are included here to save space.

Proposition 1. *Algorithm BuildDOGMAIndex correctly builds a DOGMA index for an RDF graph $G_{\mathcal{R}}$. Moreover, the worst-case time complexity of Algorithm BuildDOGMAIndex is $O(|E_{\mathcal{R}}| + \Lambda(k)\frac{|V_{\mathcal{R}}|}{k})$ where $\Lambda(k)$ is the worst-case time complexity of Algorithm GraphPartition over a graph with k vertices and $O(k)$ edges.*

4 Algorithms for Processing Graph Queries

In this section, we first present the DOGMA_basic algorithm for answering queries against a DOGMA index stored on external memory. We then present various extensions that improve query answering performance on complex graph queries.

4.1 The DOGMA_basic Query Processing Algorithm

Fig. 4 shows our basic algorithm for answering graph matching queries using the DOGMA index. In the description of the algorithm, we assume the existence of two index retrieval functions: retrieveNeighbors($\mathbf{D}_{\mathcal{R}}, v, l$) that retrieves from DOGMA index $\mathbf{D}_{\mathcal{R}}$ the unique identifiers for all vertices v' that are connected to vertex v by an edge labeled l, i.e., the neighbors of v restricted to label l, and retrieveVertex($\mathbf{D}_{\mathcal{R}}, v$) that retrieves from $\mathbf{D}_{\mathcal{R}}$ a complete description of vertex v, i.e., its unique identifier and its associated metadata. Note that retrieveVertex implicitly exploits locality, since after looking up neighboring vertices, the probability is high that the page containing the current vertex's description is already in memory.

DOGMA_basic is a recursive, depth-first algorithm which searches the space of all substitutions for the answer set to a given query Q w.r.t an RDF database $\mathcal{R}$. For each variable vertex v in Q, the algorithm maintains a set of constant vertices $R_v \subseteq V_{\mathcal{R}}$ (called result candidates) to prune the search space; for each answer substitution θ for Q, we have $\theta(v) \in R_v$. In other words, the result candidates must be a superset of the set of all matches for v. Hence, we can prune the search space by only considering those

```
   Algorithm DOGMA_basic
   Input: Graph query Q, DOGMA index D_R, partial substitution θ, candidate sets {R_z}
   Output: Answer set A, i.e. set of substitutions θ s.t. Qθ is a subgraph of G_R
1  if ∀z ∈ V_Q ∩ VAR : ∃c : (z → c) ∈ θ
2    A ← A ∪ {θ}
3    return /* done - a correct answer substitution has been found */
4  if θ = ∅
5    for all z ∈ V_Q ∩ VAR
6      R_z ← null /* no candidate substitutions for any vars in the query initially */
7    for all c ∈ V_Q ∩ (S ∪ V)
8      for all edges e = (c, v) incident on c and some v ∈ V_Q ∩ VAR
9        if R_v = null
10         R_v ← retrieveNeighbors(D_R, c, λ_Q(e)) /* use index to retrieve all nbrs of c with same label as e */
11       else
12         R_v ← R_v ∩ retrieveNeighbors(D_R, c, λ_Q(e)) /* restrict space of possible subst. for z */
13 R_w ← argmin_{R_z ≠ null, s.t. z ∈ V_Q ∩ V \ dom(θ)} |R_z|
14 if R_w = ∅
15   return "NO"
16 else
17   for all m ∈ R_w
18     retrieveVertex(D_R, m)
19     θ' ← θ ∪ {w → m}
20     for all z ∈ V_Q ∩ VAR
21       R'_z ← R_z
22     for all edges e = (w, v) incident on w and some v ∈ V_Q ∩ VAR \ dom(θ)
23       if R_v = null
24         R'_v ← retrieveNeighbors(D_R, m, λ_Q(e))
25       else
26         R'_v ← R_v ∩ retrieveNeighbors(D_R, m, λ_Q(e))
27     DOGMA_basic(θ'(Q), D_R, θ', {R'_z})
```

Fig. 4. DOGMA_basic algorithm

substitutions θ for which $\theta(v) \in R_v$ for all variable vertices v in Q. DOGMA_basic is called initially with an empty substitution and uninitialized result candidates (lines 4-6). We use uninitialized result candidates $R_v = \mathbf{null}$ to efficiently denote $R_v = V_{\mathcal{R}}$, i.e., the fact that there are no constraints on the result candidates yet. The algorithm then initializes the result candidates for all variable vertices v in Q which are connected to a constant vertex c in Q through an edge labeled by l (lines 7-12). Here we employ the fact that any answer substitution θ must be such that $\theta(v)$ is a neighbor of c, and thus the set of all neighbors of c in $G_{\mathcal{R}}$ reachable by an edge labeled l are result candidates for v. We use the DOGMA index $\mathbf{D}_{\mathcal{R}}$ to efficiently retrieve the neighborhood of c. If v is connected to multiple constant vertices, we take the intersection of the respective constraints on the result candidates.

At each recursive invocation, the algorithm extends the given substitution and narrows down the result candidates for all remaining variable vertices correspondingly. To extend the given substitution θ, we greedily choose the variable vertex w with the smallest set of result candidates (line 13). This yields a locally optimal branching factor of the search tree since it provides the smallest number of extensions to the current substitution. In fact, if the set of result candidates is empty, then we know that θ cannot be extended to an answer substitution, and we thus directly prune the search (lines 14-15). Otherwise, we consider all the possible result candidates $m \in R_w$ for w by deriving extended substitutions θ' from θ which assign m to w (lines 17-19) and then calling DOGMA_basic recursively on θ' (line 27). Prior to this, we update the result

candidates for all remaining variable vertices (lines 20-26). By assigning the constant vertex m to w we can constrain the result candidates for all neighboring variable vertices as discussed above.

Note that our description of the algorithm assumes that edges are undirected, to simplify the presentation. Obviously, our implementation takes directionality into account and thus distinguishes between outgoing and incoming edges when determining vertex neighborhoods.

Example 3. Consider the example query and RDF database in Fig. 1. Fig. 5(a) shows the initial result candidates for each of the variable vertices $?v_1, ?v_2, ?v_3$ in boxes. After initialization, DOGMA_basic chooses the smallest set of result candidates to extend the currently empty substitution $\theta = \emptyset$. We have that $|R_{v_1}| = |R_{v_2}| = 3$; suppose R_{v_2} is chosen. We can now extend θ by assigning each of the result candidates (Bill B0045, Bill B0532, Bill B1432) to $?v_2$. Hence, we first set $\theta'(?v_2) =$ Bill B0045. This introduces a new constant vertex into the query and we thus constrain the result candidates of the two neighbor variable vertices v_1, v_3 by the "amendmentTo" and "sponsor" neighborhood of Bill B0045 respectively. The result is shown in Fig. 5(b); here we call DOGMA_basic recursively to encounter the empty result candidates for v_1. Hence we reached a dead end in our search for an answer substitution and the algorithm backtracks to try the remaining extensions for θ. Eventually, DOGMA_basic considers the extension $v_2 \rightarrow$ Bill B1432 which leads to the query answer. □

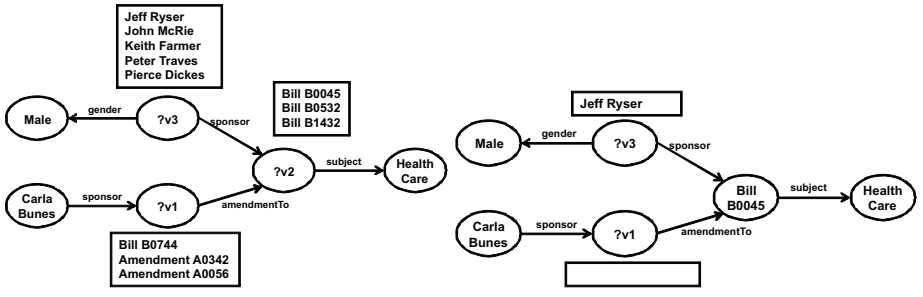

Fig. 5. Execution of DOGMA_basic on the example of Fig. 1

Proposition 2. *Suppose $\boldsymbol{D}_{\mathcal{R}}$ is a DOGMA index for an RDF database $\mathcal{R}$ and Q is a graph query. Then: DOGMA_basic$(Q, \boldsymbol{D}_{\mathcal{R}}, \{\}, null)$ returns the set of all correct answer substitutions for query Q w.r.t. $\mathcal{R}$. Moreover, the worst-case complexity of the DOGMA_basic algorithm is $O(|V_{\mathcal{R}}|^{|V_Q \cap VAR|})$.*

The algorithm is therefore exponential in the number of variables in the query in the worst case. However, the algorithm is efficient in practice as we will show in Section 5. Furthermore, we propose two extensions of the DOGMA index that improve its performance.

4.2 The DOGMA_adv Algorithm

The basic query answering algorithm presented in the previous section only uses "short range" dependencies, i.e., the immediate vertex neighborhood of variable vertices, to constrain their result candidates. While this suffices for most simple queries, considering "long range" dependencies can yield additional constraints on the result candidates and thus improve query performance. For instance, the result candidates for v_1 in our example query not only must be immediate neighbors of "Carla Bunes": in addition, they must be at most at a distance of 2 from "Health Care". More formally, let $d_{\mathcal{R}}(u, v)$ denote the length of the shortest path between two vertices $u, v \in V_{\mathcal{R}}$ in the undirected counterpart of a RDF graph $G_{\mathcal{R}}$, and let $d_Q(u, v)$ denote the distance between two vertices in the undirected counterpart of a query Q; a long range dependency on a variable vertex $v \in V_Q$ is introduced by any constant vertex $c \in V_Q$ with $d_Q(v, c) > 1$.

We can exploit long range dependencies to further constrain result candidates. Let v be a variable vertex in Q and c a constant vertex with a long range dependency on v. Then any answer substitution θ must satisfy $d_Q(v, c) \geq d_{\mathcal{R}}(\theta(v), c)$ which, in turn, means that $\{m \mid d_{\mathcal{R}}(m, c) \leq d_Q(v, c)\}$ are result candidates for v. This is the core idea of the DOGMA_adv algorithm shown in Fig. 6, which improves over and extends DOGMA_basic. In addition to the result candidates sets R_v, the algorithm maintains sets of distance constraints C_v on them. As long as a result candidates set R_v remains uninitialized, we collect all distance constraints that arise from long range dependencies on the variable vertex v in the constraints set C_v (lines 15-16 and 34-35). After the result candidates are initialized, we ensure that all elements in R_v satisfy the distance constraints in C_v (lines 17-18 and 37-38). Maintaining additional constraints therefore reduces the size of R_v and hence the number of extensions to θ we have to consider (line 23 onward).

DOGMA_adv assumes the existence of a *distance index* to efficiently look up $d_{\mathcal{R}}(u, v)$ for any pair of vertices $u, v \in V_{\mathcal{R}}$ (through function retrieveDistance), since computing graph distances at query time is clearly inefficient. But how can we build such an index? Computing all-pairs-shortest-path has a worst-case time complexity $O(|V_{\mathcal{R}}|^3)$ and space complexity $O(|V_{\mathcal{R}}|^2)$, both of which are clearly infeasible for large RDF databases. However, we do not need to know the *exact* distance between two vertices for DOGMA_adv to be correct. Since all the distance constraints in DOGMA_adv are *upper bounds* (lines 18, 31, and 38), all we need is to ensure that $\forall u, v \in V_{\mathcal{R}}$, retrieveDistance($\mathbf{D}_{\mathcal{R}}, u, v$) $\leq d_{\mathcal{R}}(u, v)$.

Thus, we can extend the DOGMA index to include distance information and build two "lower bound" distance indexes, DOGMA_ipd and DOGMA_epd, that use approximation techniques to achieve acceptable time and space complexity.

4.3 DOGMA_ipd

For building the DOGMA index, we employed a graph partitioner which minimizes cross edges, to ensure that strongly connected vertices are stored in close proximity on disk; this implies that distant vertices are likely to be assigned to distinct sets in the partition. We exploit this to extend DOGMA to a distance index.

As seen before, the leaf nodes of the DOGMA index $\mathbf{D}_{\mathcal{R}}$ are labeled by subgraphs which constitute a partition of $G_{\mathcal{R}}$. For any node $N \in \mathbf{D}_{\mathcal{R}}$, let P_N denote the union

```
   Algorithm DOGMA_adv
   Input: Graph query Q, DOGMA Index D_R, partial substitution θ, candidate sets {R_z}, constraint sets {C_z}
   Output: Answer set A, i.e. set of substitutions θ s.t. θ(Q) ⊑̃ G
1  if ∀z ∈ V_Q ∩ VAR : ∃c : (z → c) ∈ θ
2     A ← A ∪ {θ}
3     return
4  if θ = ∅
5     for all z ∈ V_Q ∩ VAR
6        R_z ← null
7     for all c ∈ V_Q ∩ (S ∪ V)
8        for all edges e = (c, v) incident on c and some v ∈ V_Q ∩ VAR
9           if R_v = null
10             R_v ← retrieveNeighbors(D_R, c, λ_Q(e))
11          else
12             R_v ← R_v ∩ retrieveNeighbors(D_R, c, λ_Q(e))
13    for all c ∈ V_Q ∩ (S ∪ V)
14       for all variable vertices v ∈ V_Q ∩ VAR s.t. d_Q(c, v) > 1
15          if R_v = null
16             C_v ← C_v ∪ {(c, d_Q(c, v))}
17          else
18             R_v ← {u ∈ R_v | retrieveDistance(D_R, c, u) ≤ d_Q(c, v)}
19 R_w ← argmin_{R_z ≠ null, s.t. z ∈ V_Q ∩ VAR \ dom(θ)} |R_z|
20 if R_w = ∅
21    return
22 else
23    for all m ∈ R_w
24       retrieveVertex(D_R, m)
25       θ' ← θ ∪ {w → m}
26       for all z ∈ V_Q ∩ VAR
27          R'_z ← R_z
28          C'_z ← C_z
29       for all edges e = (w, v) incident on w and some v ∈ V_Q ∩ VAR \ dom(θ)
30          if R_v = null
31             R'_v ← {u ∈ retrieveNeighbors(D_R, m, λ_Q(e)) | ∀(c, d) ∈ C_v : retrieveDistance(D_R, c, u) ≤ d}
32          else
33             R'_v ← R_v ∩ retrieveNeighbors(D_R, m, λ_Q(e))
34       for all variable vertices v ∈ V_Q ∩ VAR \ dom(θ) s.t. d_Q(w, v) > 1
35          if R_v = null
36             C_v ← C_v ∪ {(m, d_Q(w, z))}
37          else
38             R_v ← {w ∈ R_v | retrieveDistance(D_R, m, v) ≤ d_Q(w, v)}
39       DOGMA_basic(θ'(Q), D_R, θ', {R'_z}, {C'_z})
```

Fig. 6. DOGMA_adv algorithm

of the graphs labeling all leaf nodes reachable from N. Hence, P_N is the union of all subgraphs in $G_{\mathcal{R}}$ that were eventually merged into the graph labeling N during index construction and therefore corresponds to a larger subset of $G_{\mathcal{R}}$. For example, the dashed lines in Fig 1(a) mark the subgraphs P_N for all index tree nodes N of the DOGMA index shown in Fig. 2 where bolder lines indicate boundaries corresponding to nodes of lower depth in the tree.

The DOGMA *internal partition distance* (DOGMA_ipd) index stores, for each index node N and vertex $v \in P_N$, the distance to the outside of the subgraph corresponding to P_N. We call this the *internal partition distance* of v, N, denoted $ipd(v, N)$, which is thus defined as $ipd(v, N) = \min_{u \in V_{\mathcal{R}} \setminus P_N} d_{\mathcal{R}}(v, u)$. We compute these distances during index construction as shown in Fig. 3 (BuildTree algorithm at lines 12-14). At query time, for any two vertices $v, u \in V_{\mathcal{R}}$ we first use the DOGMA tree index to identify those distinct nodes $N \neq M$ in $\mathbf{D}_{\mathcal{R}}$ such that $v \in P_N$ and $u \in P_M$, which are at the same level of the tree and closest to the root. If such nodes do not exist (because v, u are associated with the same leaf node in $\mathbf{D}_{\mathcal{R}}$), then we set $d_{ipd}(u, v) = 0$. Otherwise we set $d_{ipd}(u, v) = \max(ipd(v, N), ipd(u, M))$. It is easy to see that d_{ipd}

is an admissible lower bound distance, since $P_N \cap P_M = \emptyset$. By choosing those distinct nodes which are closest to the root, we ensure that the considered subgraphs are as large as possible and hence $d_{ipd}(u, v)$ is the closest approximation to the actual distance.

Proposition 3. *Building the DOGMA_ipd index has a worst-case time complexity* $O(\log \frac{|V_\mathcal{R}|}{k}(|E_\mathcal{R}| + |V_\mathcal{R}| \log |V_\mathcal{R}|))$ *and space complexity* $O(|V_\mathcal{R}| \log \frac{|V_\mathcal{R}|}{k})$.

Example 4. Consider the example of Fig. 1. As shown in Fig. 7(a), there is a long range dependency between "Carla Bunes" and variable vertex v_2 at distance 2. The boldest dashed line in Fig. 1(a) marks the top level partition and separates the sets P_{N_1}, P_{N_2}, where N_1, N_2 are the two nodes directly below the root in the DOGMA index in Fig. 2. We can determine that $ipd(\text{Carla Bunes}, N_2) = 3$ and since Bill B0045 and B0532 lie in the other subgraph, it follows that $d_{ipd}(\text{Carla Bunes}, \text{B0045/B0532}) = 3$ and therefore we can prune both result candidates. □

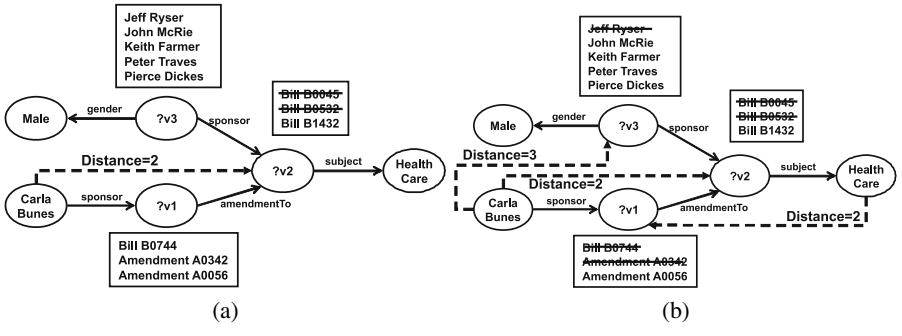

Fig. 7. Using DOGMA_ipd and DOGMA_epd for query answering

4.4 DOGMA_epd

The DOGMA *external partition distance* (DOGMA_epd) index also uses the partitions in the index tree to compute a lower bound distance. However, it considers the distance to *other* subgraphs rather than the distance within the *same* one. For some fixed level L, let $\mathcal{N}_L$ denote the set of all nodes in $\mathbf{D}_\mathcal{R}$ at distance L from the root. As discussed above, $P = \{P_N\}_{N \in \mathcal{N}_L}$ is a partition of $G_\mathcal{R}$. The idea behind DOGMA_epd is to assign a color from a fixed list of colors C to each subgraph $P_N \in P$ and to store, for each vertex $v \in V_\mathcal{R}$ and color $c \in C$, the shortest distance from v to a subgraph colored by c. We call this the *external partition distance*, denoted $epd(v, c)$, which is thus defined as $epd(v, c) = \min_{u \in P_N, \phi(P_N) = c} d_\mathcal{R}(v, u)$ where $\phi : P \to C$ is the color assignment function. We store the color of P_N with its index node N so that for a given pair of vertices u, v we can quickly retrieve the colors c_u, c_v of the subgraphs to which u and v belong. We then compute $d_{epd}(v, u) = \max(epd(v, c_u), epd(u, c_v))$. It is easy to see that d_{epd} is an admissible lower bound distance.

Ideally, we want to assign each partition a distinct color but this exceeds our storage capabilities for large database sizes. Our problem is thus to assign a limited

number of colors to the subgraphs in such a way as to maximize the distance between subgraphs of the same color. Formally, we want to minimize the objective function $\sum_{P_N \in P} \sum_{P_M \in P, \phi(P_N)=\phi(P_M)} \frac{1}{d(P_N,P_M)}$ where $d(P_N, P_M) = \min_{u \in P_N, v \in P_M} d_{\mathcal{R}}(u, v)$. Inspired by the work of Ko and Rubenstein on peer-to-peer networks [4], we designed a probabilistic, locally greedy optimization algorithm for the maximum distance coloring problem named ColorRegions, that we do not report here for reasons of space. The algorithm starts with a random color assignment and then iteratively updates the colors of individual partitions to be locally optimal. A *propagation radius* determines the neighborhood that is analyzed in determining the locally optimal color. The algorithm terminates if the cost improvement falls below a certain threshold or if a maximum number of iterations is exceeded.

Proposition 4. *Computing the external partition distance has a worst-case time complexity* $O(|C|\,(|E_{\mathcal{R}}| + |V_{\mathcal{R}}| \log |V_{\mathcal{R}}|))$ *and space complexity* $O(|V_{\mathcal{R}}|\,|C|)$.

Example 5. Consider the example of Fig. 1(a) and assume each set in the lowest level of the DOGMA index in Fig. 2 is colored with a different color. Figure 7(b) indicates some long range dependencies and shows how the external partition distance can lead to additional prunings in the three result candidates sets which can be verified against Fig. 1(a). □

5 Experimental Results

In this section we present the results of the experimental assessment we performed of the DOGMA_adv algorithm combined with DOGMA_ipd and DOGMA_epd indexes.

We compared the performance of our algorithm and indexes with 4 leading RDF database systems developed in the Semantic Web community that are most widely used and have demonstrated superior performance in previous evaluations [5]. *Sesame2* [6] is an open source RDF framework for storage, inferencing and querying of RDF data, that includes its own RDF indexing and I/O model and also supports a relational database as its storage backend. We compare against Sesame2 using its native storage model since initial experiments have shown that Sesame2's performance drops substantially when backed by a relational database system. *Jena2* [7] is a popular Java RDF framework that supports persistent RDF storage backed by a relational database system (we used PostgreSQL [8]). SPARQL queries are processed by the ARQ query engine which also supports query optimization [9]. *JenaTDB* [10] is a component of the Jena framework providing persistent storage and query optimization for large scale RDF datasets based on a native indexing and I/O model. Finally, *OWLIM* [11] is a high performance semantic repository based on the Sesame RDF database. In the experiments, we compared against the internal memory version of OWLIM which is called *SwiftOWLIM* and is freely available. SwiftOWLIM loads the entire dataset into main memory prior to query answering and therefore must be considered to have an advantage over the other systems.

Moreover, we used 3 different RDF datasets. *GovTrack* [1] consists of more than 14.5 million triples describing data about the U.S. Congress. The *Lehigh University Benchmark* (LUBM) [12] is frequently used within the Semantic Web community as the basis

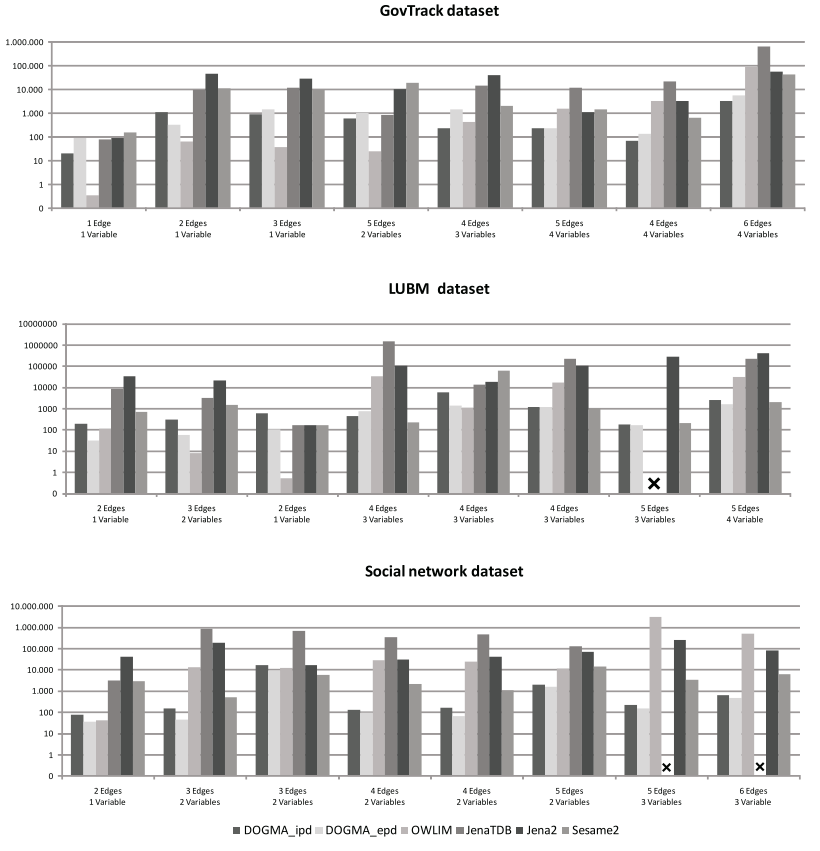

Fig. 8. Query times (ms) for graph queries of low complexity

for evaluation of RDF and ontology storage systems. The benchmark's RDF generator employs a schema which describes the university domain. We generated a dataset of more than 13.5 million triples. Finally, a fragment of the Flickr social network [13] dataset was collected by researchers of the MPI Saarbrücken to analyze online social networks [14] and was generously made available to us. The dataset contains information on the relationships between individuals and their memberships in groups. The fragment we used for the experiments was anonymized and contains approximately 16 million triples. The GovTrack and social network datasets are well connected (with the latter being denser than the former), whereas the dataset generated by the LUBM benchmark is a sparse and almost degenerate RDF graph containing a set of small and loosely connected subgraphs.

In order to allow for a meaningful comparison of query times across the different systems, we designed a set of graph queries with varying complexity, where constant vertices were chosen randomly and queries with an empty result set were filtered out. Queries were grouped into classes based on the number of edges and variable vertices.

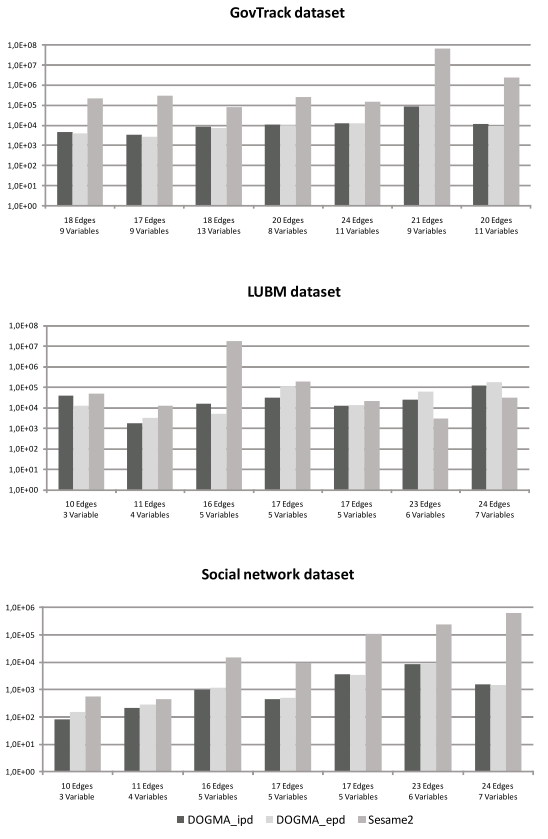

Fig. 9. Query times (ms) for graph queries of high complexity

We repeated the query time measurements multiple times for each query, eliminated outliers, and averaged the results. Finally, we averaged the query times of all queries in each class. All experiments were executed on a machine with a 2.4Ghz Intel Core 2 processor and 3GB of RAM.

In a first round of experiments, we designed several relatively simple graph queries for each dataset, containing no more than 6 edges, and grouped them into 8 classes.The results of these experiments are shown in Fig. 8 which reports the query times for each query class on each of the three datasets. Missing values in the figure indicate that the system did not terminate on the query within a reasonable amount of time (around 20 mins). Note that the query times are plotted in logarithmic scale to accommodate the large discrepancies between systems. The results show that OWLIM has low query times on low complexity queries across all datasets. This result is not surprising, as OWLIM loads all data into main memory prior to query execution. The performance advantage of DOGMA_ipd and DOGMA_epd over the other systems increases with query complexity on the GovTrack and social network dataset, where our proposed

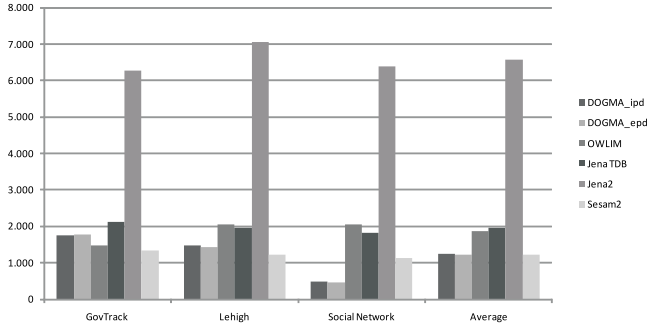

Fig. 10. Index size (MB) for different datasets

techniques are orders of magnitude faster on the most complex queries. On the LUBM dataset, however, Sesame2 performs almost equally for the more complex queries. Finally, DOGMA_epd is slightly faster on the LUBM and social network dataset, whereas DOGMA_ipd has better performance on the Govtrack dataset.

In a second round of experiments, we significantly increased the complexity of the queries, which now contained up to 24 edges. Unfortunately, the OWLIM, JenaTDB, and Jena2 systems did not manage to complete the evaluation of these queries in reasonable time, so we exclusively compared with Sesame2. The results are shown in Fig. 9. On the GovTrack and social network dataset, DOGMA_ipd and DOGMA_epd continue to have a substantial performance advantage over Sesame2 on all complex graph queries of up to 40000%. For the LUBM benchmark, the picture is less clear due to the particular structure of the generated dataset explained before.

Finally, Fig. 10 compares the storage requirements of the systems under comparison for all three datasets. The results show that DOGMA_ipd,DOGMA_epd and Sesame2 are the most memory efficient.

To wrap up the results of our experimental evaluation, we can observe that both DOGMA_ipd and DOGMA_epd are significantly faster than all other RDF database systems under comparison on complex graph queries over non-degenerate graph datasets. Moreover, they can efficiently answer complex queries on which most of the other systems do not terminate or take up to 400 times longer, while maintaining a satisfactory storage footprint. DOGMA_ipd and DOGMA_epd have similar performance, yet differences exist which suggest that each index has unique advantages for particular queries and RDF datasets. Investigating these is subject of future research.

6 Related Work

Many approaches to RDF storage have been proposed in the literature and through commercial systems. In Section 5 we briefly reviewed four such systems that we used in the performance comparison. Discussing all prior work on RDF storage and retrieval in detail is beyond the scope of this paper. Approaches differ with respect to their storage

regime, index structures, and query answering strategies. Some systems use relational databases as their back-end [15]; for instance by inferring the relational schema of the given RDF data [16,17], or using a triple based denormalized relational schema [7], whereas others propose native storage formats for RDF [11]. To efficiently retrieve triples, RDF databases typically rely on index structures, such as the popular B-tree and its generalizations, over subjects, predicates, objects or any combination thereof [18]. Query answering is either handled by the relational database back-end after a SPARQL query is translated into its SQL equivalent or employs existing index structures to retrieve stored triples that match the query. [19] does some additional tuning through B^+-tree page compression and optimized join processing. Recent work on query optimization for RDF uses triple selectivity estimation techniques similar to those used in relational database systems [9].

Despite these differences, the great majority of RDF databases are *triple oriented* in the sense that they focus on the storage and retrieval of individual triples. In contrast, our work is *graph oriented* because we analyze the graph spanned by RDF data and exploit graph properties, such as connectedness and shortest path lengths, for efficient storage and, more importantly, retrieval. This explains DOGMA's performance advantage on complex queries. GRIN [20] was the first RDF indexing system to use graph partitioning and distances in the graphs as a basis for indexing for SPARQL-like queries. However, GRIN did not operate on disk and the authors subsequently found errors in the experimental results reported in that paper. There is also some related work in other communities. LORE [21], a database system for semi-structured data, proposed path indexes based on the assumption that the input data can be accurately represented as a tree. This assumption clearly does not hold for RDF data. Furthermore, there is a lot work on approximate query answering over graph datasets in the bioinformatics community [22]. However, the biological datasets are small enough to fit into main memory and hence storage and retrieval are not being addressed. Finally, [19,23] focus on the physical data structures to optimally store RDF triples. Their work is thus orthogonal to ours, since a DOGMA index could be built on the physical data structures proposed in these papers in order to additionally exploit graph distance locality.

7 Conclusions and Future Work

In this paper, we proposed the DOGMA index for fast subgraph matching on disk and developed algorithms to answer queries over this index. The algorithms use efficient (but correct) pruning strategies and can be combined with two different extensions of the index. We tested a preliminary implementation of the proposed techniques against four existing RDF database systems, showing very good query answering performance. Future work will be devoted to an in-depth study of the advantages and disadvantages of each of the proposed indexes when dealing with particular queries and RDF datasets. Moreover, we plan to extend our indexes to support efficient updates, also trying to improve over usual index maintenance schemes such as those based on a partial use of the space in index nodes.

References

1. GovTrack dataset: http://www.govtrack.us
2. Seaborne, A., Prud'hommeaux, E.: SPARQL query language for RDF. W3C recommendation (January 2008)
3. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 20, 359–392 (1999)
4. Ko, B., Rubenstein, D.: Distributed self-stabilizing placement of replicated resources in emerging networks. Networking, IEEE/ACM Transactions on 13(3), 476–487 (2005)
5. Lee, C., Park, S., Lee, D., Lee, J., Jeong, O., Lee, S.: A comparison of ontology reasoning systems using query sequences. In: Proceedings of the 2nd international conference on Ubiquitous information management and communication, Suwon, Korea, pp. 543–546. ACM, New York (2008)
6. Sesame2: http://www.openrdf.org
7. Wilkinson, K., Sayers, C., Kuno, H., Reynolds, D.: Efficient RDF storage and retrieval in Jena2. In: Proceedings of SWDB, vol. 3, pp. 7–8 (2003)
8. PostgreSQL: http://www.postgresql.org
9. Stocker, M., Seaborne, A., Bernstein, A., Kiefer, C., Reynolds, D.: SPARQL basic graph pattern optimization using selectivity estimation. In: Proceeding of the 17th international conference on World Wide Web, Beijing, China, pp. 595–604. ACM, New York (2008)
10. JenaTDB: http://jena.hpl.hp.com/wiki/TDB
11. Kiryakov, A., Ognyanov, D., Manov, D.: OWLIM - a pragmatic semantic repository for OWL. In: WISE Workshops, pp. 182–192 (2005)
12. The Lehigh University Benchmark: http://swat.cse.lehigh.edu/projects/lubm
13. Flickr: http://www.flickr.com
14. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement, pp. 29–42. ACM, New York (2007)
15. Theoharis, Y., Christophides, V., Karvounarakis, G.: Benchmarking database representations of RDF/S Stores, pp. 685–701 (2005)
16. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: An architecture for storing and querying RDF data and schema information. In: Spinning the Semantic Web, pp. 197–222 (2003)
17. Sintek, M., Kiesel, M.: RDFBroker: A signature-based high-performance RDF store. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 363–377. Springer, Heidelberg (2006)
18. Harth, A., Decker, S.: Optimized index structures for querying RDF from the Web. In: Proceedings of the 3rd Latin American Web Congress, pp. 71–80 (2005)
19. Neumann, T., Weikum, G.: RDF-3X: a RISC-style engine for RDF. PVLDB 1(1), 647–659 (2008)
20. Udrea, O., Pugliese, A., Subrahmanian, V.S.: GRIN: A graph based RDF index. In: AAAI, pp. 1465–1470 (2007)
21. Goldman, R., McHugh, J., Widom, J.: From semistructured data to XML: migrating the Lore data model and query language. In: Proceedings of the 2nd International Workshop on the Web and Databases (WebDB 1999), pp. 25–30 (1999)
22. Tian, Y., McEachin, R.C., Santos, C.: SAGA: a subgraph matching tool for biological graphs. Bioinformatics 23(2), 232 (2007)
23. Neumann, T., Weikum, G.: Scalable join processing on very large RDF graphs. In: SIGMOD Conference, pp. 627–640 (2009)