Optimizing Query Shortcuts in RDF Databases

Vicky Dritsou[1,3], Panos Constantopoulos[1,3],
Antonios Deligiannakis[2,3], and Yannis Kotidis[1,3]

[1] Athens University of Economics and Business, Athens, Greece
[2] Technical University of Crete, Crete, Greece
[3] Digital Curation Unit, IMIS, Athena Research Centre, Greece
{vdritsou,panosc,kotidis}@aueb.gr, adeli@softnet.tuc.gr

Abstract. The emergence of the Semantic Web has led to the creation of large semantic knowledge bases, often in the form of RDF databases. Improving the performance of RDF databases necessitates the development of specialized data management techniques, such as the use of shortcuts in the place of path queries. In this paper we deal with the problem of selecting the most beneficial shortcuts that reduce the execution cost of path queries in RDF databases given a space constraint. We first demonstrate that this problem is an instance of the quadratic knapsack problem. Given the computational complexity of solving such problems, we then develop an alternative formulation based on a bi-criterion linear relaxation, which essentially seeks to minimize a weighted sum of the query cost and of the required space consumption. As we demonstrate in this paper, this relaxation leads to very efficient classes of linear programming solutions. We utilize this bi-criterion linear relaxation in an algorithm that selects a subset of shortcuts to materialize. This shortcut selection algorithm is extensively evaluated and compared with a greedy algorithm that we developed in prior work. The reported experiments show that the linear relaxation algorithm manages to significantly reduce the query execution times, while also outperforming the greedy solution.

1 Introduction

The Semantic Web involves, among other things, the development of semantic repositories in which structured data is expressed in RDF(S) or OWL. The structure of this data - and of its underlying ontologies - is commonly seen as a directed graph, with nodes representing concepts and edges representing relationships between concepts. A basic issue that semantic repositories need to address is the formulation of ontology-based queries, often by repeatedly traversing particular paths [11] of large data graphs. Regardless of the specific model used to store these data graphs (RDF/OWL files, relational databases, etc.), path expressions require substantial processing; for instance, when using relational databases, multiple join expressions are often involved [3,20,22,28].

By analogy to materialized views in relational databases, a *shortcut* construct can be used in RDF repositories to achieve better performance in formulating and executing frequent path queries. In this paper we elaborate on the creation of *shortcuts* that correspond to frequently accessed paths by augmenting the schema and the data graph of an

G. Antoniou et al. (Eds.): ESWC 2011, Part II, LNCS 6644, pp. 77–92, 2011.

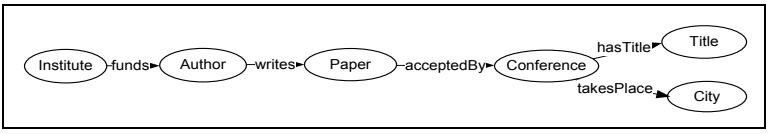

Fig. 1. Sample Schema Graph

RDF repository with additional triples, effectively substituting the execution of the corresponding path queries. Considering all possible shortcuts in a large RDF repository gives rise to an optimization problem in which we seek to select shortcuts that maximize the reduction of query processing cost subject to a given space allocation for storing the shortcuts. For this problem, which turns out to be a knapsack problem, known to be NP-hard, we have previously developed a greedy algorithm [13] which, however, leans on the side of wasting space whenever it comes to dealing with particular types of query workloads, especially correlated ones. We thus consider an alternative formulation in this paper, which leads to the development of a more efficient linear algorithm.

The contributions of this work are: (i) we elaborate on the notion of shortcuts and the decomposition of a set of user queries that describe popular user requests into a set of simple path expressions in the RDF schema graph of an RDF repository, which are then combined in order to form a set of candidate shortcuts that can help during the evaluation of the original queries; (ii) we formally define the shortcut selection problem as one of maximizing the expected benefit of reducing query processing cost by materializing shortcuts in the knowledge base, under a given space constraint; (iii) we provide an alternative formulation of the shortcut selection problem that trades off the benefit of a shortcut with the space required for storing its instances in the RDF database: the constraint matrix of the resulting bi-criterion optimization problem enjoys the property of total unimodularity, by virtue of which we obtain very efficient classes of linear programming solutions; and (iv) through extensive experimental evaluation we demonstrate that the linear algorithm outperforms our greedy solution in most cases.

2 Problem Formulation

In this section we provide the underlying concepts that are required in order to formulate our problem. Sections 2.1 and 2.2 describe these preliminary concepts, which have been introduced in [13].

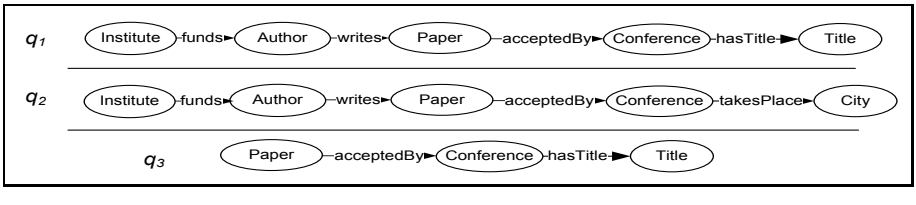

Fig. 2. Sample Queries

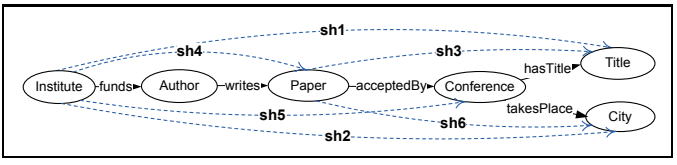

Fig. 3. Candidate Shortcuts

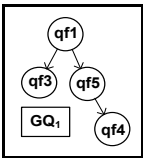

Fig. 4. Graph GQ_1

2.1 Preliminary Concepts

Assume an RDF database G containing two parts: (i) a schema graph, $G_S(V_S, E_S)$, where V_S is a set of nodes that represent entity classes (or, concepts) and a set of edges E_S, representing relationship classes (or, properties); and (ii) a data graph, $G_D(V_D, E_D)$, consisting of a set of nodes, V_D, that are instances of nodes in V_S and a set of edges, E_D, that are instances of edges in E_S. More generally, G could be any graph-structured semantic database. A sample schema graph of an RDF database is presented in Figure 1. Assume now a query workload $Q = \{q_1, q_2, q_3, \ldots, q_m\}$ of m queries. Each q_i is associated with a frequency f_i and comprises a data-level query (querying the schema graph is out of scope in this paper). Moreover, it is represented as a weakly connected subgraph G_q of the schema graph G_S. For example, consider the schema graph of Figure 1 and the three related queries: q_1: "For each institute, find the titles of the conferences in which the papers of the authors it funds have been accepted"; q_2: "For each institute, find the cities where the conferences in which the papers of the authors it funds have been accepted are taking place"; and q_3: "For each paper, find the title of the conference in which it has been accepted". The subgraphs of these queries are shown in Figure 2. Regardless of the query language used, to evaluate the queries and retrieve their matching instances we need to traverse the corresponding paths.

In this paper we deal only with path queries, like the three examples presented above. More complex queries can also be formulated, requiring more than one paths in order to be expressed, e.g. in our sample graph a query that relates $Institute$s with conference $Title$s together with the $City$ each one is located. Although this query cannot be part of our assumed workload, we can retrieve its matching instances by joining the results of the path queries it contains (namely q_1 and q_2).

Query fragments and candidate shortcuts. Each path query q_i in Q has a *length* l_i expressing the number of relationships its path contains. We call *query fragment* any subpath containing more than one edge. In our framework, a shortcut matches exactly one such query fragment. Given a query workload Q, containing $|Q|$ queries, and L the length of the query having the longest path among all queries in Q, there are $O(|Q| \times L^2)$ query fragments (and shortcuts). This large number is reduced when we define as *candidate shortcuts* only those that originate from or terminate to nodes that we consider to be "interesting": a node $u \in G_S$ is considered to be a *candidate shortcut node* iff (i) u is a starting node or an ending node of a query $q \in Q$; or (ii) u is a starting node or an ending node of two different edges e_i, e_j of G_S, each being traversed by at least one query $q \in Q$. Having defined the set of candidate nodes, we develop the set SH of *candidate shortcuts* by considering all valid combinations between candidate shortcut nodes, so that these are connected through an existing path of G_S and this path is traversed by at least one

query $q \in Q$. For example, consider again the sample graph of Figure 1 and the queries of Figure 2 constituting the workload Q. Given Q, the candidate shortcut nodes are: (i) *Institute*, as starting node of q_1 and q_2; (ii) *Paper*, as starting node of q_3; (iii) *Conference*, since two edges belonging to two different queries originate from it; (iv) *Title*, as ending node of q_1 and q_3; and (v) *City*, as ending node of q_2. Starting from these, we generate the set of candidate shortcuts presented in Figure 3[1].

Each candidate shortcut sh_i maps to exactly one query fragment qf_i, which may be contained in more than one queries of Q. The set of queries in which qf_i is contained is called *related queries* of sh_i and is denoted as RQ_i. For instance, the set of related queries of sh_4 in Figure 3 is $RQ_4 = \{q_1, q_2\}$. Regarding the relationships between shortcuts, we distinguish the following three cases: (i) a shortcut sh_i is *disjoint* to sh_j if the query fragments they map to do not share any common edge; (ii) a shortcut sh_i is *fully contained* in sh_j iff the query fragment that sh_i maps to is a subpath of the corresponding query fragment of sh_j, denoted hereafter as $sh_i \prec sh_j$; and (iii) a shortcut sh_i *overlaps* sh_j if they are not disjoint, and none of them fully contains the other. Finally, for each shortcut sh_i we denote the set $SF_i = \{qf_j \mid qf_j \prec qf_i\}$.

2.2 Estimating Shortcut Benefit

We now turn to defining the benefit of introducing a shortcut and formulating shortcut selection as a benefit maximization problem. Assume a candidate shortcut sh_i with underlying query fragment qf_i and its set of related queries RQ_i. An estimate of the cost of retrieving the answer to qf_i may be given by (i) the number of edges that need to be traversed in the data graph G_D; (ii) the actual query time of qf_i; (iii) an estimate by a query optimizer. For the formulation of our problem we assume that this cost is given by the number of edges, tr_i, that need to be traversed in the data graph G_D (however in Section 4 we present a comparison of the results obtained when considering as cost the number of traversals on one hand and the actual query times on the other). Note that not all edges traversed necessarily lead to an answer, however they do contribute to the counting of tr_i. Now suppose that we augment the database by shortcut sh_i. This involves inserting one edge in the schema graph G_S, while in the data graph G_D one edge is inserted for each result of qf_i, a total of r_i edges between the corresponding nodes. Hence, by inserting one instance of sh_i for each result of qf_i using RDF Bags[2], thus allowing duplicate instances of shortcuts (triples) to be inserted, we retain the result cardinalities and obtain exactly the same query results. Then the new cost of answering qf_i is equal to the new number of traversals required, i.e. the number r_i of results of qf_i. Therefore, the benefit obtained by introducing a shortcut sh_i in order to answer qf_i is equal to the difference between the initial cost minus the new cost, $tr_i - r_i$. Since qf_i is used in answering each of its related queries, the benefit for query $q_k \in RQ_i$ can be estimated by multiplying the fragment benefit by the frequency of the query, i.e. $f_k(tr_i - r_i)$. The total benefit obtained by introducing shortcut sh_i is then equal to the sum of the benefits obtained for each related query. If the query fragments

[1] Due to readability issues, the number contained in the label of shortcuts is not presented in the figure as subscript.

[2] http://www.w3.org/TR/rdf-schema/

underlying the candidate shortcuts are disjoint then the aggregate benefit is the sum of the benefits of all candidate shortcuts. If, however, there are containment relationships between fragments, things get more complicated. For example, take the above mentioned queries q_1 and q_3 described in Figure 2. According to the definition of candidate shortcuts, there are four candidates beneficial for these queries, namely sh_1, sh_3, sh_4 and sh_5. Assume now that shortcut sh_5 has been implemented first and that adding sh_1 is being considered. The benefit of adding sh_1 with regard to query q_1 will be smaller than what it would have been without sh_5 in place. Indeed, sh_1 is not going to eliminate the traversals initially required for the fragment $qf_1 = Institute \overset{funds}{\rightarrow} Author \overset{writes}{\rightarrow} Paper \overset{acceptedBy}{\rightarrow} Conference \overset{hasTitle}{\rightarrow} Title$ but, rather, the traversals of edges of type sh_5. We denote as d_{ij} the difference in the number of traversals required to answer qf_i due to the existence of a shortcut induced by qf_j. This difference is positive for all $qf_j \prec qf_i$, i.e. for each $qf_j \in SF_i$. Finally, shortcuts induced by overlapping query fragments do not interfere in the above sense: since the starting node of one of them is internal to the other, they cannot have common related queries.

Let us now turn to the space consumption of shortcuts. Regardless of how many queries are related to a query fragment qf_i, the space consumption that results from introducing the corresponding shortcut sh_i is r_i as above. The total space consumption of all shortcuts actually introduced should not exceed some given space budget b.

2.3 Space-Constrained Benefit Maximization

We are now ready to formulate the problem of selecting beneficial shortcuts as a benefit maximization problem. Assume an RDF graph G, a finite set of m path queries $Q = \{q_i \mid i = 1, \ldots, m\}$, $m \in \mathbb{N}$, each with a frequency f_i, a set $QF = \{qf_i \mid i = 1, \ldots, n\}$ of n query fragments, $n \geqslant m, n \in \mathbb{N}$ deriving from the set Q and a space budget $b > 0$. Our problem is then defined as:

$$max \sum_{i=1}^{n} \sum_{q_k \in RQ_i} \{f_k(tr_i - r_i)x_i - f_k \sum_{qf_j \in SF_i} d_{ij}x_ix_j\}$$

subject to

$$\sum_{i=1}^{n} r_i x_i \leqslant b$$

$$x_i = \begin{cases} 1 & \text{if shortcut } sh_i \text{ is established} \\ 0 & \text{otherwise} \end{cases}$$

$$SF_i = \{qf_j \mid qf_j \prec qf_i, qf_j \in QF\}$$

$$RQ_i = \{q_j \mid qf_i \prec q_j, q_j \in Q\}.$$

This is a 0-1 quadratic knapsack problem, known to be NP-hard [15]. Several systematic and heuristic methods have been proposed in the literature [27] to obtain approximate solutions to this problem. All computationally reasonable methods assume non-negative terms in the objective function. This is obviously not the case of our problem, since the reductions d_{ij} of the traversals are non-negative integers thus resulting in non-positive quadratic terms.

To address this problem, we have previously developed a greedy algorithm [13] that seeks to maximize the overall benefit by selecting a subset of candidate shortcuts to materialize given a space budget. The algorithm takes as input the schema and data graph, together with the query workload, finds the candidate shortcuts and then computes the benefit of each one, considering as cost the number of edge traversals. It then incrementally selects the candidate with the maximum per-unit of space benefit that fits into the remaining budget, until it reaches the defined space consumption. This algorithm succeeds in identifying beneficial shortcuts, yet it wastes space, especially when the queries of the workload are correlated. This led us to consider an alternative formulation and develop a new algorithm, as described in the following Section. A detailed comparison of the two approaches is presented in Section 4.

3 Bi-criterion Optimization

One established class of solution methods for the 0-1 quadratic knapsack problem are Lagrangean methods, including the traditional Lagrangean relaxation of the capacity constraint and a more sophisticated variant, Lagrangean decomposition [8,10,25,27]. As explained in Section 2, such techniques are inapplicable to our problem, due to the negative quadratic terms in the objective function. However, the techniques cited above, provided the inspiration for an alternative modelling approach that would avoid the combined presence of binary decision variables and a capacity constraint.

Instead of stating a space constraint, we employ a cost term for space consumption in the objective function. This means that the utility of each calculated solution will, on one hand, increase based on the benefit of the selected shortcuts but, on the other hand, it will decrease proportionally to the solution's space consumption. The factor that determines the exact penalty incurred in our objective function per space unit significantly influences the final space required by the solution of our algorithm. A small penalty per space unit used typically leads to solutions that select many shortcuts, thus consuming a lot of space. On the contrary, a large penalty per space unit used typically leads to selecting fewer shortcuts of high benefit for materialization. As we describe at the end of this section, a binary search process will help determine the right value of this parameter, so as to select a solution that respects a space budget. Unlike Lagrangean methods that penalize consumption in excess of the budget, in our formulation space consumption is penalized from the first byte. As explained below, this formulation yields an efficiently solvable linear program.

We will now describe how the constraints are derived for three important cases:

- Specifying that a candidate shortcut may be useful (or not) for evaluating specific queries that contain its corresponding query fragment.
- Specifying how to prevent containment relations for the *same* query. For the same query, whenever we consider two materialized shortcuts such that one fully contains the other, then only one of them can be used for the evaluation of the query.[3] Expressing such constraints is crucial for the quality of the obtained solution. The

[3] In this case, it is not true that the larger of the two shortcuts will always be used, as the presence of additional materialized shortcuts for the same query may result in utilizing the smaller one.

greedy algorithm may select a smaller shortcut for materialization, and then select a shortcut that fully contains the first one.
- Specifying that two overlapping shortcuts (but none of them is fully contained in the other) cannot be useful at the same time for the same query.

In the model of Section 2, the resource-oriented treatment of space employs the space budget as a parameter for controlling system behavior. In the alternative, price-oriented model presented in this section, a price coefficient on space consumption is the control parameter. A small price on space consumption has an analogous effect to setting a high space budget. By iteratively solving the price-oriented model over a range of prices we effectively determine pairs of space consumption and corresponding solutions (sets of shortcuts). So, both the resource-oriented and the price-oriented approach yield shortcut selection policies the outcomes of which will be compared in terms of query evaluation cost and space consumption in Section 4.

Another important feature of the alternative model is the explicit representation of the usage of shortcuts in evaluating queries. In the model of Section 2, decision variables x_i express whether shortcut sh_i is inserted or not. The actual usage of selected shortcuts regarding each query is not captured by these variables. In fact, the set of shortcuts that serve a given query in an optimal solution as determined by the greedy algorithm has *no* containment relationships among shortcuts and *no* overlapping shortcuts. Of course, containment and overlapping can occur between shortcuts serving different queries. For example, consider again the queries q_1, q_2 and q_3 mentioned above. The greedy algorithm will not propose to use both shortcuts sh_1 and sh_3, which have a containment relationship, to serve q_1. Even if sh_3 is selected in one step and sh_1 in a subsequent step, only sh_1 will finally be used for the execution of q_1. Similarly, sh_5 and sh_3 will not be proposed by greedy for q_1, since their query fragments overlap. If we use sh_5 then the part of q_1 that corresponds to the subgraph $Paper \overset{acceptedBy}{\rightarrow} Conference$ will be "hidden" under the new edge sh_5 and therefore shortcut sh_3 cannot be applied.

Specifying that a shortcut is useful for a specific query. We introduce additional decision variables that express the usage of a given shortcut to serve a given query. In particular, x_{ik} denotes whether shortcut sh_i is useful for the query q_k, where $x_{ik} = \{0, 1\}$ and $q_k \in RQ_i$.

The x_{ik} variables depend on each other and on x_i in various ways. Note that if a shortcut sh_i is selected to serve any query q_k (specified by $x_{ik} = 1$), then this shortcut is also selected by our algorithm for materialization (specified by $x_i = 1$). Thus:

$$x_{ik} \leqslant x_i, \quad i = 1, ..., n; k \in RQ_i \tag{1}$$

Moreover, if a shortcut has been selected for materialization, then this shortcut is useful for at least one query. Thus:

$$x_i \leqslant \sum_{k \in RQ_i} x_{ik}, \quad i = 1, ..., n \tag{2}$$

Avoiding containment. For each query q_k we construct an auxiliary directed graph GQ_k as follows: The nodes of GQ_k represent the query fragments related to q_k. If (i) a query fragment qf_i is fully contained in another fragment qf_j, and (ii) there does not

exist any qf_k ($k \neq i, j$) such that $qf_i \prec qf_k$ and $qf_k \prec qf_j$, then an edge is inserted in GQ_k starting from the bigger fragment qf_j and ending at the smaller fragment qf_i. Note that we connect each such node with its children only and not with all of its descendants. An example illustrating such a graph is shown in Figure 4, where the graph of query q_1 is presented. This graph is constructed with the help of Figure 3, and by recalling that each candidate shortcut sh_i matches the query fragment qf_i.

Using GQ_k we generate the containment constraints by the following procedure:

1. For each root node of the graph (i.e., each node with no incoming edges) of query q_k, find all possible paths p_t^k leading to leaf nodes of the graph (i.e., to nodes with no outgoing edges) and store them in a set of paths P^k.
2. For each path $p_t^k \in P^k$, create a containment constraint by restricting the sum of all variables x_{ik} whose fragment is present in the path to be less than or equal to 1.

The second step enforces that among all the candidate shortcuts participating in each path p_t^k, at most one can be selected. Continuing our previous example based on Figure 4, there exist two paths in P^1 for query q_1, namely $p_1^1 = \{qf_1, qf_3\}$ and $p_2^1 = \{qf_1, qf_5, qf_4\}$. These paths generate two containment constraints, namely $x_{11}+x_{31} \leqslant 1$ and $x_{11} + x_{51} + x_{41} \leqslant 1$.

Avoiding overlaps. Overlapping candidates are treated in a similar way. In graph GQ_k we compare each node with all other nodes on different paths of the graph. For each pair of compared nodes, if the corresponding query fragments of the nodes under comparison have edges in common, then we insert this pair of nodes in a set OF_k which stores the pairs of overlapping fragments with respect to query q_k. For each pair in OF_k we create a constraint specifying that the sum of its decision variables be less than or equal to 1, indicating that only one of the candidate shortcuts in the pair can be selected to serve the query q_k. In the previous example of query q_1, we would check the two pairs (qf_5, qf_3) and (qf_4, qf_3). Only the first of these pairs contains fragments that have edges in common, as qf_5 and qf_3 both contain the edge $Paper \overset{acceptedBy}{\rightarrow} Conference$. In order to specify that the shortcuts of these two query fragments cannot be useful for the same query q_1, we generate the constraint: $x_{51} + x_{31} \leqslant 1$.

In conclusion, given a parameter c (called *price coefficient*) that specifies the per space penalty of the chosen solution, we obtain the following bi-criterion 0-1 integer linear programming formulation for the shortcut selection problem:

$$max \sum_{i=1}^{n} \sum_{q_k \in RQ_i} f_k(tr_i - r_i)x_{ik} - c\sum_{i=1}^{n} r_i x_i \tag{3}$$

subject to

$$x_{ik} - x_i \leqslant 0, \quad i = 1, ..., n; k \in RQ_i \tag{4}$$

$$-\sum_{k \in RQ_i} x_{ik} + x_i \leqslant 0, \quad i = 1, ..., n \tag{5}$$

$$\sum_{i \in p_u^k} x_{ik} \leqslant 1, \quad k = 1, ..., m; p_u^k \in P^k \tag{6}$$

$$x_{ik} + x_{jk} \leqslant 1, \quad k = 1, ..., m; (qf_i, qf_j) \in OF_k \tag{7}$$

$$x_{ik} = \begin{cases} 1 & \text{if shortcut } sh_i \text{ is selected for query } q_k \\ 0 & \text{otherwise} \end{cases}$$

$$x_i \in \{0, 1\}$$

3.1 Linear Relaxation

In the general case (i.e., unless particular conditions are satisfied), integer linear programs are NP-hard. In our problem though, a particular linear relaxation allows us to obtain fast solutions. Constraint (5) ensures that if *all* $x_{ik} = 0$, then also $x_i = 0$. If we remove this constraint, then it is possible to have $x_i = 1$ (i.e., to consume space) without using the shortcut in any query q_k. However, since this is a maximization problem and x_i has a negative coefficient in the objective function (3), a solution with $x_i = 1$ and all corresponding $x_{ik} = 0$ can be improved by setting $x_i = 0$. We can therefore drop the constraint from the problem, since it is never violated by optimal solutions.

For a reason that will shortly become clear, we also drop the overlap constraint(7). Contrary to constraint (5), this one can be violated by an optimal solution of the linear program. Then, a cutting plane method is used to restore feasibility with two alternative heuristics for cut selection, that will be presented in the sequel.

Consider now a relaxed linear program comprising the remaining constraints and with the 0-1 variables replaced by real variables in the interval $[0, 1]$:

$$max \sum_{i=1}^{n} \sum_{q_k \in RQ_i} f_k(tr_i - r_i)x_{ik} - c\sum_{i=1}^{n} r_i x_i \tag{8}$$

subject to

$$x_{ik} - x_i \leqslant 0, \quad i = 1, ..., n; k \in RQ_i \tag{9}$$

$$\sum_{i \in p_u^k} x_{ik} \leqslant 1, \quad k = 1, ..., m; p_u^k \in P^k \tag{10}$$

$$0 \leqslant x_{ik} \leqslant 1, \quad 0 \leqslant x_i \leqslant 1$$

The constraint matrix of the above linear program can be proven to fulfill the sufficient conditions of *total unimodularity* given in [31]. The proof is omitted due to space limitations. By virtue of this property, the linear program is guaranteed to have integer solutions (thus each x_{ik} and x_i will either be equal 0 or 1). Furthermore, the relaxation enables much more efficient solution since the empirical average performance of simplex-based linear program solvers is polynomial, in fact almost linear [4,7].

Satisfying the overlap constraints. The solution of the relaxed problem may violate constraint (9) that we have ignored in the relaxation. Here we propose a cutting plane method with two alternative heuristics for restoring constraint satisfaction. After solving the relaxed problem, we check whether the solution violates any of the omitted overlap constraints. Recall that these constraints contain two variables (representing

two shortcuts used for the same query) that are not both allowed to be equal to 1. For each such violation we set the value of one of the variables equal to 0 (i.e., we exclude the respective shortcut) and insert this as an additional constraint into the initial problem. It can be proven that adding such constraints in the problem does not affect total unimodularity. The problem is solved again and new cuts are introduced as needed until we reach a solution that does not violate any overlap constraint.

But how do we choose which of the two variables to set equal to zero? We explore two alternative heuristics. The first heuristic (LRb) selects to prohibit the shortcut with the smallest benefit for the given query q_k. The second heuristic (LRl) chooses to prohibit the shortcut with the shortest corresponding query fragment. Both heuristics have been developed in evaluation tests and are discussed in Section 4.

Selecting the Value of the Price Coefficient c. In order to find a solution that is appropriate for a given space budget, we perform a binary search on the value of c. Both linear algorithms (i.e., heuristics) developed here take as input the space budget and then create linear models based on different values of c until they reach a solution the space consumption of which is close to (or, even better, equal to) the given budget.

4 Evaluation

In this section we present an experimental study of the two variations of the linear algorithm. We implemented LRb and LRl in C++ and compared their results with our previously developed greedy algorithm (GR) described in [13]. Our goal is to evaluate the reduction in query costs w.r.t. different RDF stores and data sets, to compare the performance of LRb/LRl with GR in general and more specifically to check whether LRb/LRl can capture the dependencies produced by strongly correlated workloads in a more effective way than GR. By strongly correlated we mean workloads containing path queries with containment relationships among the majority of them. In this spirit, we report on the reduction of the total execution time of the given query workloads after augmenting the database by the proposed shortcuts, vis-a-vis the allocated space budget. We used four different systems for our experiments, each of them following a different type of RDF store approach: (i) the SWKM[4] management system, which relies on a relational DBMS, (ii) the native Sesame[5] repository, (iii) the in-memory Sesame repository and (iv) the native RDF3X[6] system. In the following experiments, unless otherwise stated, we present in the graphs the percentage of reduction achieved in the query time of the entire workload after augmenting the system by the proposed shortcuts (y-axis) with regard to the space consumed for this augmentation, expressed as a fraction of the database size (x-axis). All reported experiments were executed on a Intel Core 2 Duo 2.33GHz PC with 4GB RAM running 32bit Windows Vista. In the reported experiments we used Lingo 9.0[7] as linear solver for LRb and LRl.

Before discussing the results obtained with strongly correlated queries, we examine the dependence of performance on the query cost metrics used. Recall from Section 2.2

[4] Available at http://139.91.183.30:9090/SWKM/
[5] Available at http://www.openrdf.org/
[6] Available at http://www.mpi-inf.mpg.de/ neumann/rdf3x/
[7] LINDO Systems, http://www.lindo.com

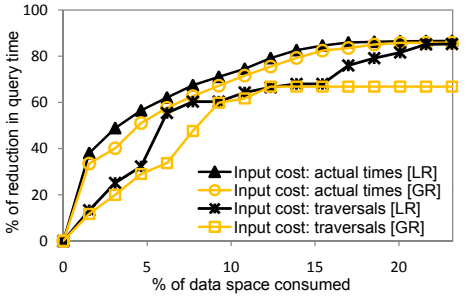

Fig. 5. Yago in Sesame Native

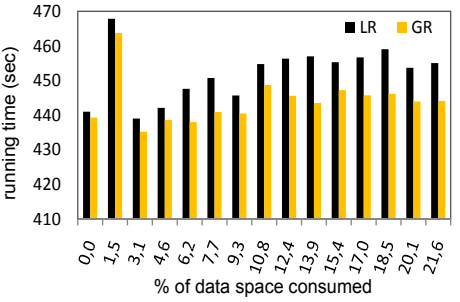

Fig. 6. Running Times of Algorithms

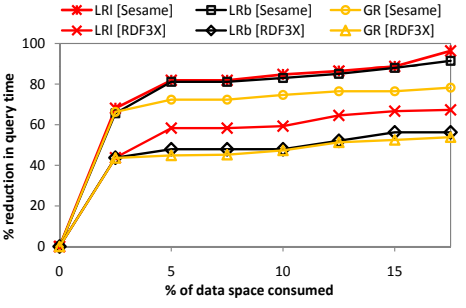

Fig. 7. Experiment with Correlated Queries

that computing the benefit of candidate shortcuts requires a cost metric of the query fragments. Our theoretical formulation employs the number of traversals required to answer the query, while the actual query times must also be considered. In [13] we have shown that both cost metrics give qualitatively similar results for the greedy algorithm. To confirm that this stands true for LRb/LRl too, we used the Yago data set [34] that contains 1.62 million real data triples. We considered as query workload the 233 distinct query paths of length 3 contained in the schema graph of Yago and used as RDF store the Sesame native repository. In Figure 5 we present the percentage of reduction in query time achieved for the entire workload (y-axis) after augmenting the system by the selected shortcuts w.r.t. the fraction of data space required to be consumed by the shortcut edges. In this experiment LRb and LRl give the same solutions, due to the small length of queries (presented as LR in Figure 5).

It is obvious that our approach significantly reduces the overall query execution time without consuming large amounts of space. Moreover, LR achieves a bigger reduction compared to greedy (GR): when using traversals as cost metric, LR reduces the query time by 55% by consuming only 6% of the data space, while GR reaches a reduction of 33% for the same consumption. LR reaches the maximum reduction of 86% of the initial query time with GR stopping at a maximum reduction of 66%. The reported results confirm that the two cost metrics used give qualitatively similar results. We

Table 1. Relative Improvement of Query Times in Sesame (over Greedy)

Space	$\frac{t_{GR}-t_{LRb}}{t_{GR}}$	$\frac{t_{GR}-t_{LRl}}{t_{GR}}$
2.5%	−2%	5%
5.0%	32%	34%
7.5%	32%	34%
10.0%	33%	40%
12.5%	36%	42%
15.0%	49%	52%
17.5%	61%	83%

will therefore continue in the following experiments by considering the traversals as input cost metric for our algorithms. Regarding the running times of the algorithms, we present in Figure 6 the time in seconds required by each algorithm to choose the appropriate shortcuts for the experiment described with the Yago data set. The graph shows the running time of the algorithms for each given budget. GR runs faster than LR in all cases, since LR must iterate through the binary search, with LR being from 2 to 13 seconds slower. On the other hand, running times are a secondary consideration, since shortcut selection is an offline procedure.

To evaluate the performance of the algorithms with strongly correlated queries, we generated a synthetic schema graph comprising 143 nodes and 117 edges and then used a Perl script to generate synthetic data. This script randomly picks a schema node at the beginning and generates one data node from it. It then produces all the data edges emanating from the latter according to the schema graph. This process is repeated on the newly created ending data nodes of the produced edges. A parameter p defining the probability of breaking a path (stop following the edges of the current node and continue with a new one) is also used, which in this experiment is set to 0.4. The data set generated for this experiment contains 1 million triples. For the correlated workload we used 31 path queries of length from 5 to 15 with 29 of them being fully contained in at least one other query of the workload; the smallest having length equal to 5 are fully contained in 10 longer queries. We used Sesame native repository and RDF3X for this experiment and the results obtained are presented in Figure 7. All algorithms show significant reduction in query time: by consuming just 2.5% of the data space they all reduce the query time by 65% in Sesame, while in RDF3X they all reach a reduction of 43% for the same space consumption. Moreover, LRb and LRl show much better results than GR in Sesame, while in RDF3X LRl outperforms the two remaining algorithms. We can thus confirm that the linear approach, and mostly LRl, captures the dependencies that exist in correlated workloads in a more effective way: LRl manages to reduce the query time by 96% when consuming only 17.5% of the data space in Sesame, while for the same consumption it achieves a reduction of 68% in RDF3X. Table 1 shows the relative improvement in query time reduction achieved in Sesame by the two variants of LR over GR for various space budgets. Both LRb and LRl show better results in most cases (LRb is by 2% worse than GR in one case of very small budget), while LRl yields 83% more efficient shortcuts than GR for the maximum budget.

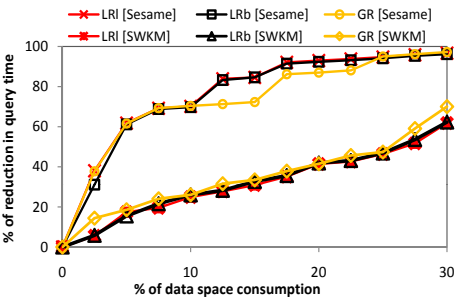

Fig. 8. Experiment with CIDOC Ontology

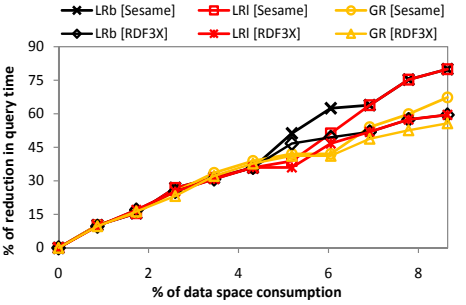

Fig. 9. Using Long Path Queries

In Figure 8 we present the results obtained by a different type of ontology, namely the CIDOC CRM[8], which is characterized by a schema graph forming a "constellation graph" structure with non-uniform density. This contains a small number of "star" nodes, each connected with a large number of "planet" nodes and sparsely connected with other star nodes. Our goal here is to evaluate the performance of LR algorithms with these schema graphs. We generated 1 million synthetic triples with the aforementioned Perl script and used as query workload 65 queries of length 2 to 4. We tested the algorithms using both the Sesame in-memory repository and the SWKM system. The reduction achieved is much bigger with Sesame than SWKM. Moreover, LRb and LRl perform better than GR when using Sesame, while this is not the case with SWKM: although in the majority of cases the three algorithms show similar results, in 3 cases GR achieves a bigger reduction by $5-7\%$.

To examine the performance of the algorithms with longer path queries, we generated a synthetic schema graph containing 189 nodes and 467 edges and populated it using the Perl script with 4 million of synthetic triples. The query workload here comprises 26 queries with length from 10 to 15. For this experiment we used the Sesame native repository and RDF3X, and obtained the results presented in Figure 9. The reduction achieved in both systems is similar in this case: all algorithms achieve a reduction of at least 40% by consuming only 5% of the data space. Moreover, LRb and LRl still give better results when compared to GR. E.g., for the maximum budget given in the experiment, the execution of the workload after augmenting the RDF3X system by the shortcuts proposed by LRl is by 39% faster than after inserting those proposed by GR.

5 Related Work

The emerging Semantic Web necessitates the development of methods to efficiently manage semantic data, often expressed in OWL and/or RDF. To this end, a variety of systems have been developed to handle RDF data including, among others, Jena[9], Sesame, RDF3X, SWKM. For the efficient querying of semantic data, several RDF

[8] Available at http://www.cidoc-crm.org/rdfs/cidoc_crm_v5.0.2_english_label.rdfs
[9] Available at http://jena.sourceforge.net/

languages have been proposed, with SPARQL [2], RQL [19] and RDQL [1] standing as main representatives.

In the relational world, views have long been explored in database management systems [30], either as pure programs [32,33], derived data that can be further queried [23], pure data (detached from the view query) or pure index [29]. The selection of shortcuts in RDF databases is similar to materialized views in relational systems, which has long been studied: selecting and materializing a proper set of views with respect to the query workload and the available system resources optimizes frequent computations [18,21]. Moreover, restricting the solution space into a set of views with containment properties has been proposed in the Data Cube [16] framework.

Indexing RDF data is similar in concept (due to the hierarchical nature of path queries) to indexing data in object oriented (OO) and XML Databases. An overview of indexing techniques for OO databases is presented in [5]. In [17] a uniform indexing scheme, termed U-index, for OO databases is presented, while in [6] the authors present techniques for selecting the optimal index configuration in OO databases when only a single path is considered and without taking into account overlaps of subpaths. The work of [12] targets building indices over XML data. Unlike XML data, RDF data is not rooted at a single node. Moreover, the above techniques target the creation of indices over paths, thus not tackling the problem of this paper, which is the selection of shortcuts to materialize given a workload of *overlapping* queries.

Following the experience from relational databases, indexing techniques for RDF databases have already been explored [14,24,35]. A technique for selecting which (out of all available) indices can help accelerate a given SPARQL query is proposed in [9]. In [3] the authors propose a vertically partitioned approach to storing RDF triples and consider materialized path expression joins for improving performance. While their approach is similar in spirit to the notion of shortcuts we introduce, the path expression joins they consider are hand-crafted (thus, there is no automated way to generate them in an arbitrary graph). Moreover the proposed solutions are tailored to relational backends (and more precisely column stores), while in our work we do not assume a particular storage model for the RDF database. Our technique can be applied in conjunction with these approaches to accelerate query processing, since it targets the problem of determining the proper set of shortcuts to materialize without assuming a particular RDF storage model. In [26] the authors propose a novel architecture for indexing and querying RDF data. While their system (RDF3X) efficiently handles large data sets using indices and optimizes the execution cost of small path queries, our evaluation shows that our technique further reduces the execution cost of long path queries in RDF3X.

6 Conclusions

Shortcuts are introduced as a device for facilitating the expression and accelerating the execution of frequent RDF path queries, much like materialized database views, but with distinctive traits and novel features. Shortcut selection is initially formulated as a benefit maximization problem under a space budget constraint. In order to overcome the shortcomings of a previous greedy algorithm solution to this problem, we developed an alternative bi-criterion optimization model and a linear solution method.

We have shown through an extensive experimental study that the augmentation of RDF databases with shortcuts reduces significantly the query time of the workload, while in the majority of cases the two variations of our algorithm outperform our previous greedy solution. This reduction was obtained using different types of RDF stores, uniform and non-uniform schema graphs, real as well as synthetic ontologies, different database sizes and different types of query workloads. In the case of strongly correlated query workloads, where the greedy solution tends to waste space, we have seen that our two variations (LRl and LRb) better capture the dependencies among queries and achieve a bigger reduction in query execution time.

References

1. RDQL - A Query language for RDF. W3C Member, `http://www.w3.org/Submission/RDQL/`
2. SPARQL Query Language for RDF. W3C Recommendation, `http://www.w3.org/TR/rdf-sparql-query/`
3. Abadi, D.J., Marcus, A., Madden, S., Hollenbach, K.J.: Scalable Semantic Web Data Management Using Vertical Partitioning. In: VLDB (2007)
4. Beasley, J.E.: Advances in Linear and Integer Programming. Oxford Science (1996)
5. Bertino, E.: A Survey of Indexing Techniques for Object-Oriented Database Management Systems. Query Processing for Advanced Database Systems (1994)
6. Bertino, E.: Index Configuration in Object-Oriented Databases. The VLDB Journal 3(3) (1994)
7. Borgwardt, K.H.: The average number of pivot steps required by the simplex-method is polynomial. Mathematical Methods of Operations Research 26(1), 157–177 (1982)
8. Caprara, A., Pisinger, D., Toth, P.: Exact Solution of the Quadratic Knapsack Problem. INFORMS J. on Computing 11(2), 125–137 (1999)
9. Castillo, R., Leser, U., Rothe, C.: RDFMatView: Indexing RDF Data for SPARQL Queries. Tech. rep., Humboldt University (2010)
10. Chaillou, P., Hansen, P., Mahieu, Y.: Best network flow bounds for the quadratic knapsack problem. Lecture Notes in Mathematics, vol. 1403, pp. 225–235 (2006)
11. Constantopoulos, P., Dritsou, V., Foustoucos, E.: Developing query patterns. In: Agosti, M., Borbinha, J., Kapidakis, S., Papatheodorou, C., Tsakonas, G. (eds.) ECDL 2009. LNCS, vol. 5714, pp. 119–124. Springer, Heidelberg (2009)
12. Cooper, B.F., Sample, N., Franklin, M.J., Hjaltason, G.R., Shadmon, M.: A Fast Index for Semistructured Data. In: VLDB (2001)
13. Dritsou, V., Constantopoulos, P., Deligiannakis, A., Kotidis, Y.: Shortcut selection in RDF databases. In: ICDE Workshops. IEEE Computer Society, Los Alamitos (2011)
14. Fletcher, G.H.L., Beck, P.W.: Indexing social semantic data. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, Springer, Heidelberg (2008)
15. Gallo, G., Hammer, P., Simeone, B.: Quadratic knapsack problems. Mathematical Programming 12, 132–149 (1980)
16. Gray, J., Bosworth, A., Layman, A., Pirahesh, H.: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. In: ICDE (1996)
17. Gudes, E.: A Uniform Indexing Scheme for Object-Oriented Databases. Information Systems 22(4) (1997)
18. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing Data Cubes Efficiently. In: SIGMOD Conference (1996)

19. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A declarative query language for RDF. In: WWW (2002)
20. Kotidis, Y.: Extending the Data Warehouse for Service Provisioning Data. Data Knowl. Eng. 59(3) (2006)
21. Kotidis, Y., Roussopoulos, N.: A Case for Dynamic View Management. ACM Trans. Database Syst. 26(4) (2001)
22. Larson, P., Deshpande, V.: A File Structure Supporting Traversal Recursion. In: SIGMOD Conference (1989)
23. Larson, P., Yang, H.Z.: Computing Queries from Derived Relations. In: VLDB (1985)
24. Liu, B., Hu, B.: Path Queries Based RDF Index. In: SKG, Washington, DC, USA (2005)
25. Michelon, P., Veilleux, L.: Lagrangean methods for the 0-1 Quadratic Knapsack Problem. European Journal of Operational Research 92(2), 326–341 (1996)
26. Neumann, T., Weikum, G.: The rdf-3x engine for scalable management of rdf data. VLDB J. 19(1) (2010)
27. Pisinger, D.: The quadratic knapsack problem - a survey. Discrete Applied Mathematics 155(5), 623–648 (2007)
28. Rosenthal, A., Heiler, S., Dayal, U., Manola, F.: Traversal Recursion: A Practical Approach to Supporting Recursive Applications. In: SIGMOD Conference (1986)
29. Roussopoulos, N., Chen, C.M., Kelley, S., Delis, A., Papakonstantinou, Y.: The ADMS Project: View R Us. IEEE Data Eng. Bull. 18(2) (1995)
30. Roussopoulos, N.: Materialized Views and Data Warehouses. SIGMOD Record 27 (1997)
31. Schrijver, A.: Theory of linear and integer programming. John Wiley, Chichester (1998)
32. Sellis, T.K.: Efficiently Supporting Procedures in Relational Database Systems. In: SIGMOD Conference (1987)
33. Stonebraker, M.: Implementation of Integrity Constraints and Views by Query Modification. In: SIGMOD Conference (1975)
34. Suchanek, F.M., Kasneci, G., Weikum, G.: Yago: A Core of Semantic Knowledge. In: WWW. ACM Press, New York (2007)
35. Udrea, O., Pugliese, A., Subrahmanian, V.S.: GRIN: A Graph Based RDF Index. In: AAAI (2007)