A Similarity Measure for Approximate Querying over RDF data

Roberto De Virgilio
Università Roma Tre
Rome, Italy
dvr@dia.uniroma3.it

Antonio Maccioni
Università Roma Tre
Rome, Italy
maccioni@dia.uniroma3.it

Riccardo Torlone
Università Roma Tre
Rome, Italy
torlone@dia.uniroma3.it

ABSTRACT

Approximate query answering relies on a similarity measure that evaluates the relevance, for a given query, of a set of data extracted from the underlying database. In the context of graph-modeled data, many methods (such as, subgraph isomorphism, graph edit distance, and maximum common subgraph) have been proposed to face this problem. Unfortunately, they are usually hard to compute and when they are used on RDF data, several drawbacks arise. In this paper, we propose a measure to evaluate the similarity between a (small) graph representing a query and a portion of a (large) graph representing an RDF data set. We show that this measure: (i) can be evaluated in linear time with respect to the size of the given graphs and, (ii) guarantees other interesting properties. In order to show the feasibility of our approach, we have used such similarity measure in a technique for approximate query answering. The technique has been implemented in a prototypical system and a number of experimental results obtained with this system confirm the effectiveness of the proposed measure.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Query Processing; H.3.3 [**Information Search and Retrieval**]: Search process

General Terms

Algorithms, Performance

Keywords

Top-k, Graph Matching, Monotonicity, RDF

1. INTRODUCTION

The Semantic Web technology and the Linked Open Data initiative are fostering the generation and availability of a large quantity of RDF data. This phenomenon is turning the Web into a global knowledge base, where resources are identified by means of URIs, semantically described by RDF, and related through RDF statements. Indeed, RDF is the "de-facto" standard language for the representation of semantic information: it encodes Web data as a labeled directed graph in which the nodes represent the resources and links represent semantic relationships between resources. Queries over RDF data can also be expressed as graph and, basically, query answering consists in finding the portion of the data graph that matches with the given query graph,

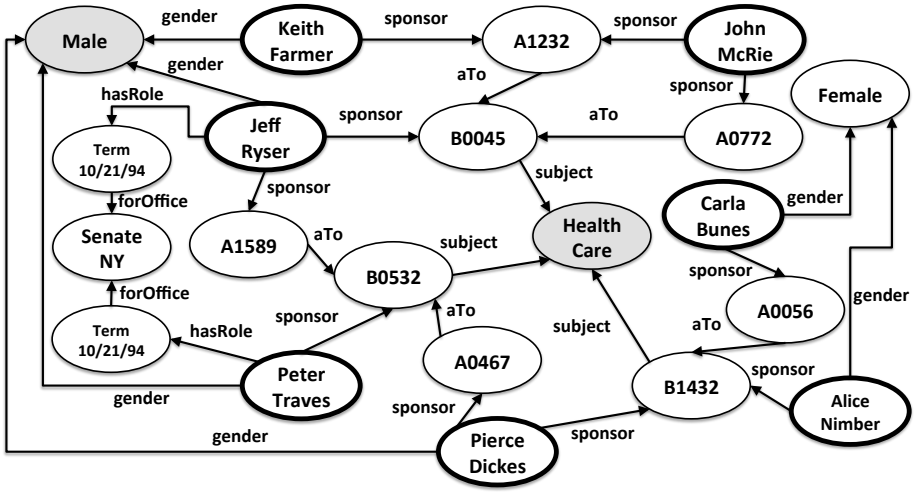
Let us consider for instance the RDF graph G_d depicted in Figure 1, taken from [2]: it represents a simplified portion of the GovTrack[1] , a database that stores events that occurred in the US Congress. The graph query Q_1 asks for all amendments ($?v1$) sponsored by Carla Bunes to a bill ($?v2$) on the subject of Health Care that was originally sponsored by a male person ($?v3$).

In this framework, the rapid increase of data availability has raised a number of data management issues [5, 6]. Among them, a major problem lies in the difficulty for users to specify queries that retrieve the information they really need. As a result, approximate query processing is increasingly capturing the attention of researchers [2, 8, 10, 24, 29] since they relax the matching relationship between queries and data, and thus provide an effective support to non-expert users, who are usually unaware of the way in which data is organized. For instance, if we relax query answering over RDF data, the same answer of Q_1 can be returned to the query Q_2 reported in Figure 1, for which there is indeed no exact answer.

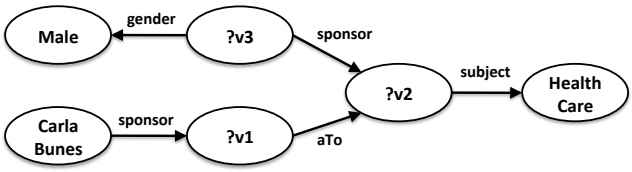
This problem is often tackled using graph-based methods such as subgraph isomorphism, graph edit distance, and maximum common subgraph, which however are known to be NP-hard problems [12, 17, 18]. Therefore, many approximate approaches to query answering on graph shaped data have been proposed to reduce the complexity of the problem [2, 28, 29, 10, 26]. They rely on heuristics [26], on the use of specific indexing structures [2, 28, 29], and on fixing some threshold on the maximum number of *hops* (i.e. node/edge additions/deletions needed to perfectly match the query graph with the underlying graph database) that are allowed [10]. All these methods work well on biological and chemical data but are usually not suited for semantic and social data, where noise is often present and the organization is totally different [18]. For this reason, specific methods for finding similarities and patterns over graph data have been proposed [15, 18]. Still, the high computational complexity makes these methods unfeasible in many practical situations.

[1] http://www.govtrack.us

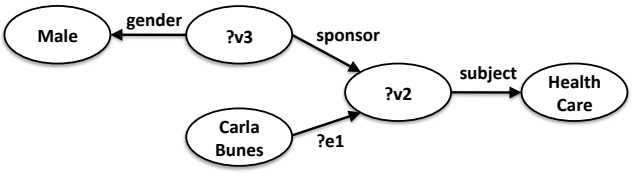
EDBT/ICDT '13, March 18 - 22, 2013, Genoa, Italy.



(a) A RDF data graph G_d



(b) A query Q_1



(c) A query Q_2

Figure 1: An example of data and query graph

In this paper, we propose a new measure of similarity for RDF data that approximates the notion of graph edit distance and can be computed in linear time with respect to the size of the input. This measure is used in a novel technique for approximate query answering over RDF data based on the simple observation that different paths of a query graph usually represent different semantic relationships between nodes. For instance, the edges of Q_1 in the above example say that Male is the gender of someone sponsoring something on the subject Health Care. It follows that query answering can proceed as follows: first, the query is decomposed into a set of paths that start from a source and end into a sink, then those paths are matched against the data graph, and finally the data paths that best match the query paths are combined to generate the answer. With a suitable relaxation of the notion of alignment between graph paths and data paths, we can adopt the same strategy to generate approximate answers to queries. Consequently, our similarity measure basically accounts for the degree of alignment of two paths in a graph.

In order to test the feasibility of our approach, we have developed a system[2] for querying RDF data that implements the above mentioned technique for query answering. Experiments over widely used benchmarks have shown that our technique outperforms other approaches, in terms of both effectiveness and efficiency.

The rest of the paper is organized as follows. In Section 2 we discuss related work. In Section 3 we introduce some preliminary notions and definitions. In Sections 4 we illustrate our similarity distance with its properties, in Section 5 we show an approach supported by the metric that solves the approximate query answering over RDF. The experimental results on the effectiveness of the metric for such algorithms are shown in Section 6 and finally, in Section 7, we draw some conclusions and sketch some future work.

2. RELATED WORK

Many research efforts have focused on graph similarities, specially from the field of graph matching [12]. In fact, a first category of works relies on subgraph isomorphism [30]. However the well-known intractability of the problem inspired approximate approaches to simplify the problem [12, 9]. In particular, graph simulation techniques has been used to make graph matching tractable. A second category of works focuses on the adoption of special indexes. In particular, several approaches have proposed in-memory structures for indexing the nodes of the data graph [23], while others have proposed specific indexes for the efficient execution of SPARQL queries and joins [20]. In addition, other proposals tackle the problem by indexing graph substructures (e.g., paths, frequent subgraphs, trees). Typically, these indexes are exploited in problems dealing with graph matching, to filter out graphs that do not match the input query. Approaches in this area can be classified in graph indexing and subgraph indexing. In graph indexing approaches, such as gIndex [25], TreePi [27], and FG-Index [4], the graph database consists of a set of small graphs. The indexing aims at finding all the database graphs that contain or are contained in a given query graph. On the other hand, subgraph indexing approaches, such as DOGMA [2], TALE [22], GADDI [28], SAPPER [29], and Zeng et al. [26] aim at indexing large database graph, with the goal of finding efficiently all (or a subset of) the subgraphs that match a given

[2] A prototype application is available at https://www.dropbox.com/sh/d5u1u24qnyqg18f/7Oefq8-qVa

query. Finally, there are works on reachability [16, 21] and distance queries [3] based on testing the existence of a path between two nodes in the graph and on the evaluation of the distance between them. An interesting approach is proposed in [10] where the authors reformulate the query graph in terms of a bounded query in which an edge denotes the connectivity of nodes within a predefined number of hops. This guarantees a cubic time complexity for the graph matching problem.

Most of the mentioned works are focused on medical, chemical and proteinic networks and they are usually not efficient over semantic and social data [18]. Therefore, specialized metrics where proposed [15, 18]. GMO [15] introduces a structural metric based on a bipartite graph, NESS [18] proposes a measure based on both topological and content information in the neighborhood of a node of the graph. All these approaches differ quite a lot from our method. Indeed, we tackle the problem using a technique that takes into account the structural constraints on how different relations between nodes have to be correlated. It relies on the tractable problem of alignment between paths.

3. PRELIMINARY ISSUES

This section states some preliminary definitions useful to introduce the notion of similarity.

3.1 Basics

RDF is the standard language for the representation of semantic information: it encodes Web data as a labeled directed graph in which the nodes represent resources and links represent semantic relationships between them. A resource can be either a Web entity identified by a URI or a value (also called literal). Given a set $\mathcal{U}$ of *URIs* and a set $\mathcal{L}$ of *literals*, let us assume a set $\Sigma_N = \mathcal{U} \cup \mathcal{L}$ of node labels and a set $\Sigma_E = \mathcal{U}$ of edge labels. RDF data can be formally represented as a labelled directed graph G as follows.

Definition 1 (Data Graph). *A data graph $G = < N, E, L_N, L_E >$ is a labelled directed graph where N is a set of nodes, $E \subseteq N \times N$ is a set of ordered pairs of nodes, called edges, and L_N and L_E are labeling functions associating an element of Σ_N to each node in N and an element of Σ_E to each edge in E, respectively.*

Let `VAR` be a set of *variables*, denoted by the prefix "?", a *query graph* Q is defined as follows.

Definition 2 (Query Graph). *A query graph Q is a data graph where $\Sigma_N = \mathcal{U} \cup \mathcal{L} \cup$ `VAR` and $\Sigma_E = \mathcal{U} \cup$ `VAR`.*

A *substitution* for a query graph Q is a function that maps the variables in Q to either URIs or literals. A *transformation* τ on a query graph is a sequence of the following basic update operations: node and edge insertion, node and edge deletion, and labeling modification of both nodes and edges.

Definition 3 (Query answer). *An (approximate) answer to a query graph Q over a data graph G is a subgraph G' of G for which there exists a substitution ϕ and a transformation τ such that $G' = \tau(\phi(Q))$. If τ is empty, G' is an* exact *answer to Q.*

Intuitively, an answer $a_1 = \tau_1(\phi_1(Q))$ is more relevant than another answer $a_2 = \tau_2(\phi_2(Q))$ if a_1 is more similar to Q than a_2, that is, τ_1 contains a lower number of operations than τ_2. In the context of RDF data in which nodes represent concepts and edges represent relationships, it is useful to associate a weight of relevance to each basic update operation. For instance, it is reasonable, in the RDF domain, that the modification of a label is less relevant than a node insertion, since the latter increases the semantic distance between concepts. Therefore, let ω be a function that associates a *weight of relevance* to each basic operation $\odot$. We say that the *cost* γ of a transformation $\tau = \odot_1 \circ \ldots \circ \odot_z$ is $\gamma(\tau) = z \cdot \sum_{i=1}^{z}(\omega(\odot_i))$.

Definition 4 (Relevance of an answer). *An answer $a_1 = \tau_1(\phi_1(Q))$ is more relevant than another answer $a_2 = \tau_2(\phi_2(Q))$ if $\gamma(\tau_1) < \gamma(\tau_2)$.*

This notion generalizes the definitions of graph edit distance and graph isomorphism. In fact, if $a_1 = \tau_1(\phi_1(Q))$ is more relevant than another answer $a_2 = \tau_2(\phi_2(Q))$, it means that: (i) a_1 is more similar to Q than a_2, (ii) the graph edit distance between a_1 and Q is lower than the graph edit distance between a_2 and Q, and also that (iii) a_1 is more isomorphic to Q than a_2. We will show in the next section that our measure for RDF is coherent with this definition.

3.2 Path Alignment

In RDF data, different paths denote different relationships between nodes. For instance, the edges of Q_1 indicate that Male is the gender of someone sponsoring something on the subject Health Care. Similarly, we can explain the relationships in the paths of the data graph of Figure 1(c). As usual, in a graph a node is a *source* if it has no incoming edges, while it is a *sink* if it has no outcoming edges. A *path* in a graph is a sequence of labels from a source to a sink.

Definition 5 (path). *Given a data graph $G = \{N, E, L_N, L_E\}$, a path is a sequence $l_{n_1} - l_{e_1} - l_{n_2} - \ldots - l_{e_{k-1}} - l_{n_k}$ where $l_{n_i} = L_N(n_i)$, $l_{e_i} = L_E(e_i)$, $n_i \in N$, $e_i \in E$, n_1 is a source and n_k is a sink.*

Sources are used as starting points for navigating the graph through paths. If sources are not present in the given data graph, we promote the *hub* nodes to play this role. A node is a hub in a data graph G if the difference between the number of outgoing edges and the number of the incoming edges is maximum in G.

The data graph in Figure 1 has seven sources (the double-marked nodes) and two sinks (*Health Care* and *Male*, marked in gray). An example of path is:

$$p_z = \text{JR-sponsor-A1589-aTo-B0532-subject-HC}$$

where JR and HC denote *Jeff Ryser* and *Health Care*, respectively. The length of a path is the number of nodes occurring in the path, while the position of a node corresponds to its position in the path. For instance, p_z has length 4 and the node A1589 has position 2. The query Q_1 in Figure 1 has the following paths:

q_1 : CB-sponsor-?v1-aTo-?v2-subject-HC
q_2 : ?v3-sponsor-?v2-subject-HC
q_3 : ?v3-gender-Male

Our measure evaluates the similarity between an RDF graph a_i and another RDF graph Q by applying substitutions and transformations to the paths of Q. This operation is called alignment.

DEFINITION 6 (ALIGNMENT). *Given an answer a_i and a query graph Q an alignment is a substitution ϕ and a transformation τ of a path p of Q such that $\tau(\phi(p))$ is a path of a_i.*

Intuitively, we will calculate the similarity distance by computing alignments on the paths of the graphs.

4. A SIMILARITY MEASURE

In this section we define the RDF similarity distance, called *score*, and we show how it is coherent with the notion of relevance given in the previous section.

4.1 Scoring function

The function *score* is an approximate implementation of the general notion of relevance (Definition 4) that can be computed in linear time against the size of the data to assess. The function *score* simulates the relevance of answers a_i by taking into account two different aspects, *quality* and *conformity*. The former measures how much the paths retrieved align with the paths in the query. The latter measures how much, in a_i, the combination of paths retrieved is similar to the combination of the paths in the query.

The first aspect that *score* considers is the quality of alignment between paths of an answer a_i and paths of a query Q as follows:

$$\Lambda(a_i, Q) = \sum_{q \in Q} (\lambda(p, q))$$

In this formula, q is a path of Q, p is the path of a_i that originates from an alignment $\tau \circ \phi$ of q (that is, $p = \tau(\phi(q))$), and λ is a function defined as follows:

$$\lambda(p, q) = (a \cdot n_N^- + b \cdot n_N^{\curvearrowright}) + c \cdot n_E^- + d \cdot n_E^{\curvearrowright}) \quad (1)$$

In this expression: (i) n_N^- and n_E^- are, respectively, the number of nodes and edges of p that are not present in q, and (ii) $n_N^{\curvearrowright}$ and $n_E^{\curvearrowright}$ are, respectively, the number of nodes and edges inserted in q by τ. Finally, a, b, c and d are parameters that serve to take into account the weights of relevance of the operations in τ (see Definition 4).

The second aspect that the *score* function considers is the conformity between the combination of the paths in the solution and the combination of the paths in the query. This is evaluated as follows:

$$\Psi(a_i, Q) = \sum_{q_i, q_j \in Q} (\psi(q_i, q_j, p_i, p_j))$$

In this equation, q_i and q_j are paths of Q, p_i and p_j are paths of a_i that originate from alignments $\tau_i \circ \phi_i$ and $\tau_j \circ \phi_j$ of q_i and q_j respectively (that is, $p_i = \tau_i(\phi_i(q_i))$ and $p_j = \tau_j(\phi_j(q_j))$), and ψ is a function defined as follows:

$$\psi(q_i, q_j, p_i, p_j) = \begin{cases} e \cdot \frac{|\chi(q_i, q_j)|}{|\chi(p_i, p_j)|}, & \text{if } |\chi(p_i, p_j)| > 0 \\ e \cdot |\chi(q_i, q_j)|, & \text{if } |\chi(p_i, p_j)| = 0 \end{cases}$$

where χ is a function that associates with each pair of paths (p_1,p_2) the set of nodes in common between p_1 and p_2. It follows that $\psi(q_i, q_j, p_i, p_j)$ returns the ratio between the sizes of $\chi(p_i, p_j)$ and $\chi(q_i, q_j)$. Finally, e is a parameter that serves to take into account the weight of the conformity in *score*. This is very useful in those applications where the topology and the interconnections within the solutions are very important (e.g., in social network analysis), sometimes even more than the content of the data themselves.

The final score function is then computed as:

$$score(a_i, Q) = \Lambda(a_i, Q) + \Psi(a_i, Q)$$

4.2 Score and relevance

It turns out that, with a suitable choice of the parameters in Equation 1 that considers the weights of relevance assigned to the basic operations, *score* is *coherent* with the notion of relevance of an answer, that is, for each pair of answers a_1 and a_2 for a query Q such that a_1 is more relevant than a_2 we have that $score(a_1, Q) < score(a_2, Q)$.

THEOREM 1. *Given a query graph Q and a data graph G, for each pair of answers a_i and a_j for Q over G, if a_i is more relevant than a_j then we have $score(a_i, Q) < score(a_j, Q)$.*

Proof. Let $\odot_N^{\curvearrowright}$, $\odot_N^-$ and $\odot_N^{\times}$ be basic update operations of node insertion, node deletion, and labeling modification, respectively. Analogously, $\odot_E^-, \odot_E^{\curvearrowright}$ and $\odot_E^{\times}$ are the respective operations on edges. On these operations, we fix the function ω: (i) $\omega(\odot_N^-) = a$, (ii) $\omega(\odot_N^{\curvearrowright}) = b$, (iii) $\omega(\odot_E^-) = c$ and (iv) $\omega(\odot_E^{\curvearrowright}) = d$. We consider, as in other works [18], $\omega(\odot_N^{\times}) = 0$ and $\omega(\odot_E^{\times}) = 0$ because we do not want to penalize the case where the answer gathers more labels than Q.

Now, let us count the number of basic update operations in a transformation τ_i for an answer a_i. In this case: n_N^- and n_E^- are, respectively, the number of nodes and edges of a_i that are inserted in Q, and $n_N^{\curvearrowright}$ and $n_E^{\curvearrowright}$ are, respectively, the number of nodes and edges updated in Q by τ_i.

The cost of $\gamma(\tau_i)$ is $n_N^- \cdot a + n_N^{\curvearrowright} \cdot b + n_E^- \cdot c + n_E^{\curvearrowright} \cdot d$. Let $a_1 = \tau_1(\phi_1(Q))$ and $a_2 = \tau_2(\phi_2(Q))$ be two answers over a query Q. Considering, from Definition 4, that $a_1 = \odot_1^1 \circ \ldots \circ \odot_z^1$ is more relevant than $a_2 = \odot_1^2 \circ \ldots \circ \odot_y^2$, we have that

$$\gamma(\tau_1) < \gamma(\tau_2) \quad (2)$$

But for a path $p \in a_i$ we have that $\gamma(\tau_i) = \lambda(p, Q)$. Then, if we generalize Equation 2 to all the paths of the two answers a_1 and a_2 we obtain

$$\Lambda(a_1, Q) < \Lambda(a_2, Q) \quad (3)$$

that satisfies the hypothesis for the first aspect of *score*. With *score* we have an upper bound of the notion of relevance because if a node has a mismatch in Q and it is in common among more than one path, then it gets counted more than once in n_N^- and $n_N^{\curvearrowright}$. The conformity $\Psi(a_i, Q)$ follows a similar trend than $\Lambda(a_i, Q)$. In fact, more are the mismatching elements (i.e. nodes and edges) and lower is the number of common nodes. Consequently, the number of intersections between the paths in an answer conforming to the intersections between the paths of the query is also lower. In our case we have that

$$\Psi(a_1, Q) < \Psi(a_2, Q). \quad (4)$$

Given Equation 3 and Equation 4, it follows that $score(a_1, Q) < score(a_2, Q)$.

□

4.3 Computation of Alignment

In order to measure Λ in *score* we have to compute alignments between paths in a and paths in Q. a and Q are decomposed into a set of paths that start from a source and end into a sink. Then the paths of a are aligned against paths of Q. In our example, this method would decompose Q_1 in the following paths:

q_1 : Carla Bunes $\xrightarrow{sponsor}$?v1 $\xrightarrow{aTo}$?v2 $\xrightarrow{subject}$ Health Care
q_2 : ?v3 $\xrightarrow{sponsor}$?v2 $\xrightarrow{subject}$ Health Care
q_3 : ?v3 $\xrightarrow{gender}$ Male

Now imagine to have a_1 extracted from G_d that is formed by the following paths:

p_1 : Carla Bunes $\xrightarrow{sponsor}$ A0056 $\xrightarrow{aTo}$ B1432 $\xrightarrow{subject}$ Health Care
p_2 : Pierce Dickens $\xrightarrow{sponsor}$ B1432 $\xrightarrow{subject}$ Health Care
p_3 : Pierce Dickens $\xrightarrow{gender}$ Male

Note that in the example above the result is an exact answer to Q_1, but the same strategy can be adopted to compare paths that are not exactly matches and thus, with a lower degree of similarity.

We can now compute the quality of alignments between the paths p in a_1 and the paths q in Q. They are done by inserting, deleting and modifying nodes in q by proceeding with a scan contrary to the direction of the edges. For instance let us consider q_1 and q_2 from the query graph Q_1 and the path

$$p = \text{CB-sponsor-A0056-aTo-B1432-subject-HC}$$

from G_d. We evaluate the score of p with respect to both q_1 and q_2 as follows

q_1 : CB-sponsor-?v1-aTo-?v2-subjec-HC
$\tau_1(\phi_1(q_1))$: **CB**-sponsor-**A0056**-aTo-**B1432**-subject-HC
q_2 : ?v3-sponsor-?v2-subject-HC
$\tau_2(\phi_2(q_2))$: **CB**-sponsor-**A0056**-**aTo-B1432**-subject-HC

In this case q_1 requires only a substitution ϕ on the variables while q_2 employs a transformation τ to insert **aTo-B1432** and a substitution ϕ on the variables. In the former case we have $\lambda(p, q_1) = (0+0)+(0+0) = 0$, since $n_N^- = n_N^{\curvearrowright} = n_E^- = n_E^{\curvearrowright} = 0$. In the latter case $\lambda(p, q_2) = (0+b)+(0+d)$, since $n_N^- = n_E^- = 0$, and $n_N^{\curvearrowright} = n_E^{\curvearrowright} = 1$. If we set $b = 0.5$ and $d = 1$, we have $\lambda(p, q_2) = 1.5$ (i.e. p has the best alignment with q_1). In the same way, given

$$p' = \text{JR-sponsor-A1589-aTo-B0532-subject-HC}$$

we can calculate $\lambda(p', q_1) = (a+0)+(0+0)$, since $n_V^- = 1$ due to the mismatch between CB and JR. If we set $a = 1$, $\lambda(p', q_1) = 1$ (i.e. q_1 has a better alignment with p than p').

It is straightforward to demonstrate that the time complexity of the alignment is $O(I)$ where $I = |p| + |q|$ is the sum of the nodes and edges of the paths in p and q.

5. QUERY ANSWERING

In this section we summarize our procedure for approximate query answering over RDF graphs. Further details can be found in [7].

We exploit *score* to define an algorithm for approximate query answering over RDF. Given a data graph G and a query graph Q, we aim at finding the top-k answers $a_1, \ldots, a_k$ to Q according to their relevance.

Figure 2: An example of intersection query graph.

The approach is composed of two main phases: the *indexing* (done off-line), in which all the paths of G are indexed, and the *query processing* (done on-the-fly), where the query evaluation takes place. The first task will be described in more detail in Section 6.

In the second phase, all paths PD for Q are retrieved in G by exploiting the index and the best solutions are generated from PD by adopting a strategy that guarantees a polynomial time complexity with respect to the size of PD. This task is performed by the following main steps:

Preprocessing. Given a query graph Q, in this step the set PQ of all paths is computed on the fly by traversing Q from each source to any sinks. We exploit an optimized implementation of the *Breadth-first search* (BFS) traversal. The elements of PQ are organized in the the so-called *intersection query graph* (IG).

The nodes of IG are the paths of Q, while an edge (q_i, q_j) means that q_i and q_j have nodes in common. For instance, referring to Figure 1, PQ consists of the following paths.

q_1 : Carla Bunes-sponsor-?v1-aTo-?v2-subject-Health Care
q_2 : ?v3-sponsor-?v2-subject-Health Care
q_3 : ?v3-gender-Male

The intersection query graph built from q_1, q_2 and q_3 is depicted in Figure 2. For example, this data structure keeps track of the fact that q_1 and q_2 have nodes in common, i.e. ?v2 and Health Care, as q_2 and q_3 have nodes in common, i.e. only ?v3.

Clustering. In the second step we build a cluster for each element q of PQ. Then, we group in the same cluster all the paths p of G having a sink that matches the sink of q. If q provides a variable in place of the sink, we retrieve the first (constant) value v occurring in q (w.r.t. the end of q, i.e. in the contrary way) and we group in the same cluster all the paths p of G containing a label matching v. Before the insertion of a path p in the cluster for q, we evaluate the alignment needed to obtain p from q. This allows us to compute the score of p, i.e. $\lambda(p, q)$. The paths in a clusters are ordered according to their score with the greater coming first. Note that the same path p can be inserted in different clusters, possibly with a different score. As an example, given the data graph G_d and the query graph Q_1 of Figure 1, we obtain the clusters shown in Figure 3. In this case clusters cl_1, cl_2 and cl_3 correspond to the paths q_1, q_2 and q_3 of PQ, respectively; note the scores at the right side of each path and in particular the path p_1 occurring in both cl_1 and cl_2 with different scores, i.e. 0 in cl_1 and 1.5 in cl_2.

Search. The last step aims at generating the most relevant solutions by combining the paths in the clusters built

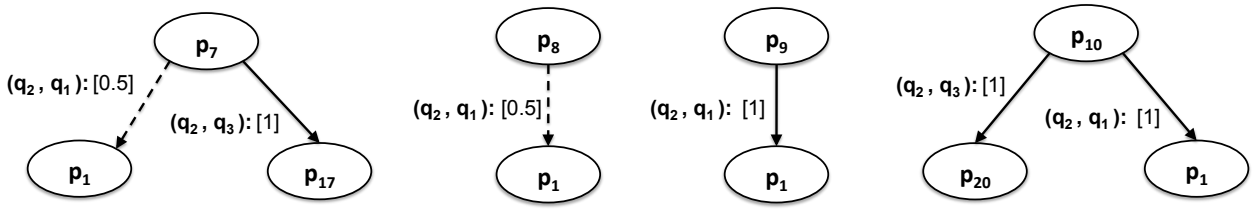


Figure 4: Forest of paths.

$$cl_1 : \begin{pmatrix} p_1 : & \text{CB-sponsor-A0056-aTo-B1432-subject-HC } [0] \\ p_2 : & \text{JR-sponsor-A1589-aTo-B0532-subject-HC } [1] \\ p_3 : & \text{KF-sponsor-A1232-aTo-B0045-subject-HC } [1] \\ p_4 : & \text{JM-sponsor-A0772-aTo-B0045-subject-HC } [1] \\ p_5 : & \text{JM-sponsor-A1232-aTo-B0045-subject-HC } [1] \\ p_6 : & \text{PD-sponsor-A0467-aTo-B0532-subject-HC } [1] \end{pmatrix}$$

$$cl_2 : \begin{pmatrix} p_7 : & \text{JR-sponsor-B0045-subject-HC } [0] \\ p_8 : & \text{PT-sponsor-B0532-subject-HC } [0] \\ p_9 : & \text{AN-sponsor-B1432-subject-HC } [0] \\ p_{10} : & \text{PD-sponsor-B1432-subject-HC } [0] \\ p_{11} : & \text{CB-sponsor-A0056-aTo-B1432-subject-HC } [1.5] \\ p_{12} : & \text{JR-sponsor-A1589-aTo-B0532-subject-HC } [1.5] \\ p_{13} : & \text{KF-sponsor-A1232-aTo-B0045-subject-HC } [1.5] \\ p_{14} : & \text{JM-sponsor-A0772-aTo-B0045-subject-HC } [1.5] \\ p_{15} : & \text{JM-sponsor-A1232-aTo-B0045-subject-HC } [1.5] \\ p_{16} : & \text{PD-sponsor-A0467-aTo-B0532-subject-HC } [1.5] \end{pmatrix}$$

$$cl_3 : \begin{pmatrix} p_{17} : & \text{JR-gender-Male } [0] \\ p_{18} : & \text{KF-gender-Male } [0] \\ p_{19} : & \text{JM-gender-Male } [0] \\ p_{20} : & \text{PD-gender-Male } [0] \end{pmatrix}$$

Figure 3: An example of the clustering step.

in the previous step. This is done by picking and combining the paths with greatest score from each cluster. The intersection query graph allows us to verify efficiently if they form a solution. As an example, given the cluster in Figure 3, the first solution is obtained by combining the paths p_1, p_{10} and p_{20} that are the elements with the greatest score in each corresponding cluster and provide the best alignment with the paths of PQ associated to the clusters.

The most tricky task of the whole process occurs in the third step above. Here, we aim at generating directly the top-k solutions by trying to minimize the number of combinations between paths. This is done by organizing the combinations of paths in a forest where nodes represent the retrieved paths, while edges between paths means that they have nodes in common. The label of each edge (p_i, p_j) is $\langle (q_i, q_j) : [\psi(q_i, q_j, p_i, p_j)] \rangle$ where q_i and q_j are the paths corresponding to the clusters where p_i and p_j were included, respectively.

For instance, Figure 4 reports the forest for the paths with the higher score extracted from the clusters in Figure 3. The label on the edge (p_{10}, p_1) indicates that if p_{10} and p_1 originate from q_2 and q_1, respectively, then $\psi(q_2, q_1, p_{10}, p_1)$ is 1. Conversely, the label on the edge (p_7, p_1) indicates that $\psi(q_2, q_1, p_7, p_1)$ is 0.5. Note that in the forest the edge (p_7, p_1) is dashed since the label is not 1. The tree in the forest with nodes p_1, p_{10} and p_{20} yields the first solution.

Such strategy exhibits, in the worst case, a quadratic time complexity w.r.t. the number of nodes of the data graph, that is $O(h \times I^2)$, where I is the number of paths retrieved by the index and h is the depth of the query graph Q (see [7] for a complete demonstration). In Section 6, experiments will show how our technique scales seamlessly with the size of the input.

6. EXPERIMENTAL RESULTS

We implemented our approach in SAMA[3] , a Java system with a Web front end. We have compared SAMA with three representatives graph matching systems: SAPPER [29], BOUNDED [10] and DOGMA [2]. Experiments were conducted on a dual core 2.66GHz Intel Xeon, running Linux RedHat, with 4 GB of memory and a 2-disk 1Tbyte striped RAID array.

6.1 Indexing

To build solutions efficiently, we index the following information: vertices' and edges' labels of the data graph G (for element-to-element mapping) and the paths ending into sinks, since they bring information that might match the query. The first information enables to locate vertices and edges matching the labels of the query graph, the second allows us to skip the expensive graph traversal at runtime. The indexing process is composed of three steps: (i) hashing of all vertices' and edges' labels, (ii) identification of sources and sinks, and (iii) computation of the paths. The first and the second step are relatively easy. The third step requires to traverse the graph starting from the sources and following the routes to the sinks. We have implemented an optimized version of the Breadth-First-Search (BFS) paradigm, where independently concurrent traversals are started from each source. Similarly to [2], and differently from the majority of related works (e.g [10]), we assume that the graph cannot fit in memory and that can only be stored on disk. Specifically, we store the index in a GraphDB, that is HyperGraphDB[4] (HGDB) v. 1.1: it models data in terms of hypergraphs. Let us recall an hypergraph H is a generalization of a graph, where an edge can connect any number of vertices. Formally $H = (X, E)$ where X is a set of nodes or vertices, and E is a set of non-empty subsets of X, called *hyperedges*. In other words, E is a subset of the power set of X. This representation allows us to define indexes on both vertices and hyperedges: $X = \{x_m | m \in M\}$ and $E = \{e_f | f \in F, e_f \subseteq X\}$,

[3] A prototype application is available at https://www.dropbox.com/sh/d5u1u24qnyqg18f/7Oefq8-qVa

[4] http://www.kobrix.com/hgdb.jsp

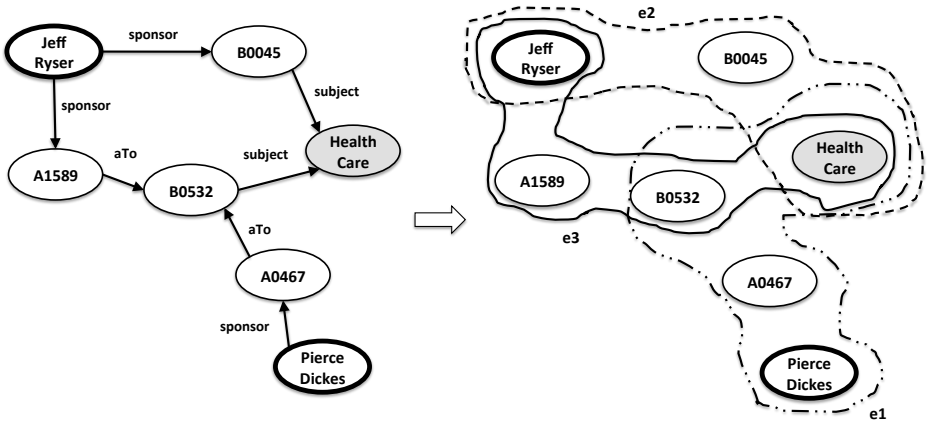


Figure 5: An example to represent a data graph G (left side) in a hypergraph H (right side)

Table 1: HyperGraphDB indexing

DG	#Triples	|HV|	|HE|	t	Space
PBlog	50K	1,5K	96K	1 sec	56 MB
GOV	1M	280K	330K	4 min	340 MB
KEGG	1M	300K	606K	7 min	700 MB
Berlin	1M	320K	700K	10 min	910 MB
IMDB	6M	900K	3M	47 min	1,2 GB
LUBM	12M	1M	15M	102 min	12,9 GB
UOBM	12M	1M	15M	102 min	12,9 GB
DBLP	26M	4M	17M	441 min	23,6 GB

where each vertex x_m and edge e_f are indexed by an index $m \in M$ and $f \in F$, respectively. Figure 5 shows an example of reference.

The matching is supported by standard IR engines (c.f. Lucene Domain index (LDi)[5]) embedded into HGDB. In particular we define a LDi index on the labels of nodes and edges. In this way, given a label, HGDB retrieves all paths containing data elements matching the label in a very efficient way (i.e. exploiting the cursors). Further, semantically similar entries such as synonyms, hyponyms and hypernyms are extracted from WordNet [11], supported by LDi.

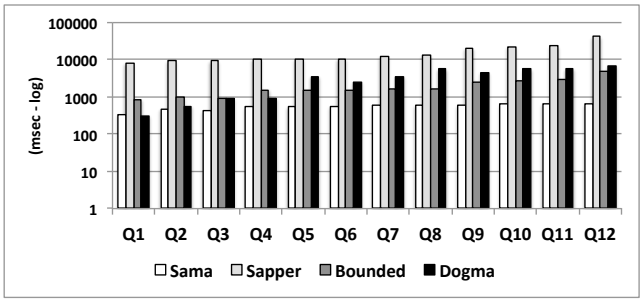
In our experiments we consider real RDF datasets, such as PBlog[6] , GovTrack, KEGG, IMDB [14], DBLP, and synthetic datasets, such as Berlin [1], LUBM [13] and UOBM [19]. Table 1 provides importing information for any dataset: number of triples, number of nodes ($|HV|$) and number of generated hyperedges ($|HE|$) in HGDB, time to create the index on HGDB (t) and memory consumption on disk. In our case, building the index takes hours for large RDF data graphs, due to the demanding traversal on the complete large graph, and requires GB of memory resources on disk to store data and metadata.

6.2 Query Execution

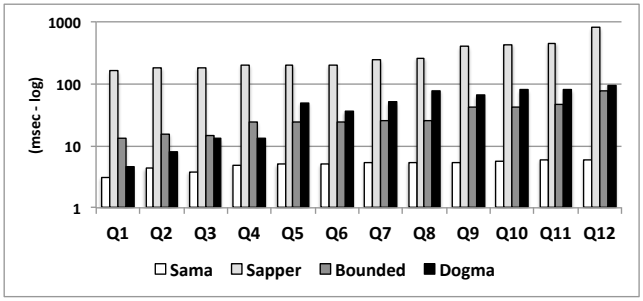
In this experiment, for each indexed dataset we formulated 12 queries in SPARQL of different complexities (i.e. number of nodes, edges and variables).

[5] http://lucene.apache.org/

[6] http://www-personal.umich.edu/~mejn/netdata/



(a) cold-cache

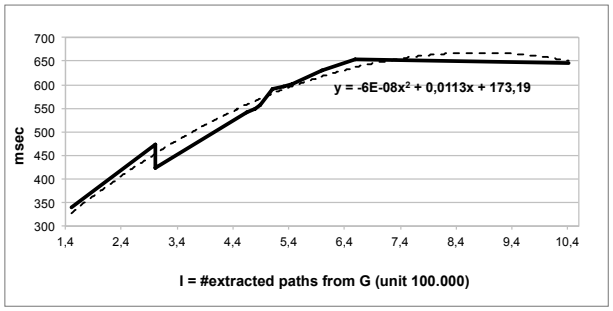


(b) warm-cache

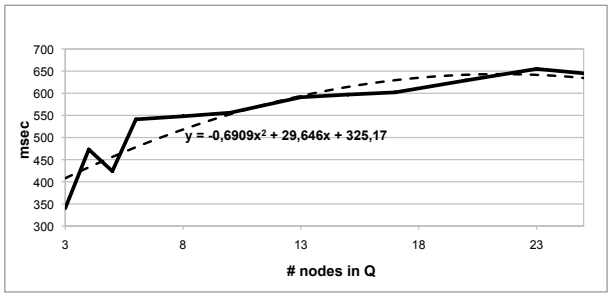
Figure 6: Average response time on LUBM: bars refer each system, using different gray scales, i.e. from Sama, white bars, to Dogma, black bars

We ran the queries ten times and we measured the average response time, in *ms* and logarithmic scale. Precisely, the total time of each query is the time for computing the top-10 answers, including any *preprocessing*, *execution* and *traversal*. We performed both cold-cache and warm-cache experiments. To make a comparison with the other systems, we reformulated the 12 queries by using the input format of each competitor. In Sama we set the coefficients of the scoring function as follows: $a = 1$, $b = 0.5$, $c = 2$ and $d = 1$. Due to space constraints, we cannot describe in detail results on every dataset, neither the query lists[7] . Therefore we show the behavior of all systems with respect to LUBM, the most

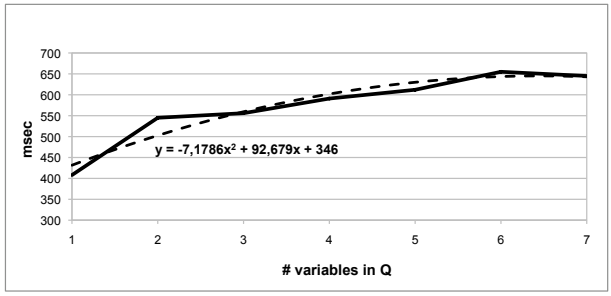
[7] At https://www.dropbox.com/sh/d5u1u24qnyqg18f/7Oefq8-qVa you can find the complete set of queries



(a)



(b)



(c)

Figure 7: Scalability of Sama on LUBM w.r.t. (a) the number I of extracted paths from G, (b) the number of nodes in Q and (c) the number of variables in Q

representative in terms of number of triples and complexity. The query run-times are shown in Figure 6.

In general Bounded performs better than Dogma, while Sapper is less efficient. Sama performs very well with respect to all competitors: it is supported by the index that retrieves all needed data elements in an efficient way (i.e. skipping data graph traversal at runtime and supporting parallel implementations).

Another aspect we test is the scalability of our approach. Such aspect is analyzed in more depth by evaluating the scalability of Sama with respect to both I and Q through distinct diagrams, as illustrated in Figure 7 (i.e. it refers to cold-cache experiments). Each diagram provides the trend-line (displaying also the associated equation): in any case the behavior of Sama is quadratic with respect to the time complexity (i.e. we have the same for warm-cache experiments).

6.3 Effectiveness

The last experiment evaluates the effectiveness of Sama

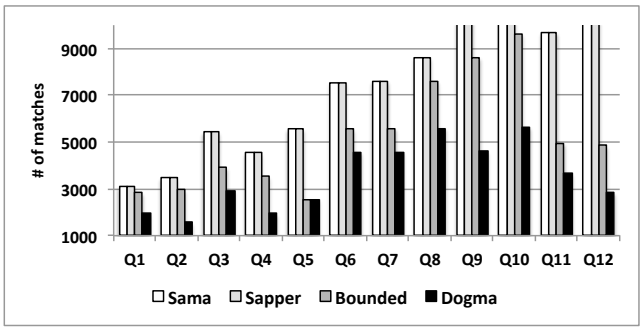


Figure 8: Effectiveness on LUBM: bars refer each system, using different gray scales, i.e. from Sama, white bars, to Dogma, black bars

and of the other competitors. The first measure we used is the reciprocal rank (RR). For a query, RR is the ratio between 1 and the rank at which the first correct answer is returned; or 0 if no correct answer is returned. In any dataset, for all 12 queries we obtained RR=1. In this case the monotonicity is never violated. To make a comparison with the other systems we inspected the matches found in terms of the solutions returned. Figure 8 shows the effectiveness of all systems on LUBM, where we run the queries without imposing the number k of solutions.

In this case Sama and Sapper always identify more meaningful matches than both Bounded and Dogma. This is due to the approximation operated by Sama and Sapper with respect to the others. We remind that the evaluation of the matches was performed by experts of the domain (e.g. LUBM). Finally, to support the meaningful of results, we measured the interpolation between precision and recall.

Figure 9 shows the results on LUBM: for Sama we depict three different trends with respect to the range of $|Q|$. As to be expected, queries with limited number of paths presents the highest quality (i.e. a precision in the range [0.5,0.8]). More complex queries decrease the quality of results, due to more data elements retrieved by the approximation, presenting good quality though. Such result confirms the feasibility of our system. The effectiveness on the other datasets follows a similar trend. On the other hand, as to be expected, the precision of the other systems dramatically decreases for large values of recall: Bounded and Dogma do not exploit an imprecise matching, while Sapper introduces *noise* (i.e. not interesting approximate results) in high values of recall.

7. CONCLUSION AND FUTURE WORK

In this paper we have presented a similarity measure for approximate querying of large RDF data sets. The approach is based on a strategy for query answering aimed at selecting and combining paths of the underlying data graph that best align with paths of the query. The similarity between queries and candidate answers is then evaluated by means of a score function that measures the degree of alignment of paths. In the worst case our technique exhibits a quadratic computational cost with respect to the size of the input and experimental results show that it behaves, in most cases, better than other approaches in terms of both efficiency and effectiveness.

This work opens several directions of further research.

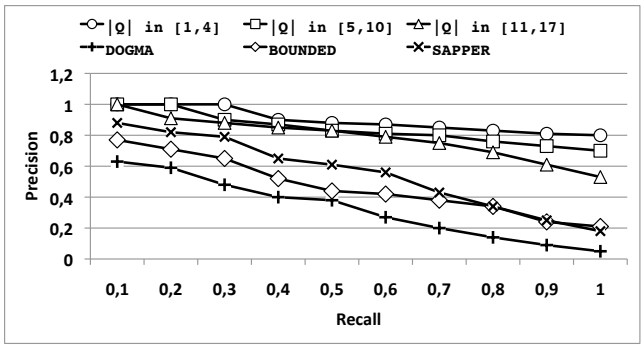


Figure 9: Effectiveness on LUBM: Precision and Recall of Sama

From a conceptual point of view, we aim to introduce improvements on the construction of answers and on the computation of the scoring function. From a practical point of view, we plan to implement the approach in a Grid environment (for instance using Hadoop/Hbase) and develop optimization techniques to speed-up the creation and the update of the index, as well as compression mechanisms for reducing the overhead required by its construction and maintenance.

8. REFERENCES

[1] C. Bizer and A. Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[2] M. Bröcheler, A. Pugliese, and V. S. Subrahmanian. Dogma: A disk-oriented graph matching algorithm for rdf databases. In *ISWC*, pages 97–113, 2009.

[3] E. P. F. Chan and H. Lim. Optimization and evaluation of shortest path queries. *VLDB J.*, 16(3):343–369, 2007.

[4] J. Cheng, Y. Ke, W. Ng, and A. Lu. Fg-index: towards verification-free query processing on graph databases. In *SIGMOD*, pages 857–872, 2007.

[5] R. De Virgilio, F. Giunchiglia, and L. Tanca, editors. *Semantic Web Information Management - A Model-Based Perspective*. Springer Verlag, 2010.

[6] R. De Virgilio, F. Guerra, and Y. Velegrakis, editors. *Semantic Search over the Web*. Springer Verlag, 2012.

[7] R. De Virgilio, A. Maccioni, and R. Torlone. Approximate querying of rdf graphs via path alignment. Technical Report RT-DIA-200, http://www.dia.uniroma3.it/Plone/ricerca/technical-reports/2012, Dipartimento di Informatica e Automazione, November 2012 - [**submitted**].

[8] R. De Virgilio, G. Orsi, L. Tanca, and R. Torlone. Nyaya: A system supporting the uniform management of large sets of semantic data. In *ICDE*, pages 1309–1312, 2012.

[9] W. Fan and P. Bohannon. Information preserving xml schema embedding. *ACM Trans. Database Syst.*, 33(1), 2008.

[10] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph pattern matching: From intractable to polynomial time. *PVLDB*, 3(1):264–275, 2010.

[11] C. Fellbaum, editor. *WordNet An Electronic Lexical Database*. The MIT Press, 1998.

[12] B. Gallagher. Matching structure and semantics : A survey on graph-based pattern matching. *Artificial Intelligence*, pages 45–53, 2006.

[13] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.

[14] O. Hassanzadeh and M. P. Consens. Linked Movie Data Base (Triplification Challenge Report). In *I-SEMANTICS*, pages 194–196, 2008.

[15] W. Hu, N. Jian, Y. Qu, and Y. Wang. Gmo: A graph matching for ontologies. In *Integrating Ontologies*, 2005.

[16] R. Jin, Y. Xiang, N. Ruan, and D. Fuhry. 3-hop: a high-compression indexing scheme for reachability query. In *SIGMOD*, pages 813–826, 2009.

[17] D. Justice and A. O. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(8):1200–1214, 2006.

[18] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *SIGMOD Conference*, pages 901–912, 2011.

[19] L. Ma, Y. Yang, Z. Qiu, G. T. Xie, Y. Pan, and S. Liu. Towards a complete owl ontology benchmark. In *ESWC*, pages 125–139, 2006.

[20] T. Neumann and G. Weikum. x-rdf-3x: Fast querying, high update rates, and consistency for rdf databases. *PVLDB*, 3(1):256–263, 2010.

[21] A. Poulovassilis and P. T. Wood. Combining approximation and relaxation in semantic web path queries. In *ISWC*, pages 631–646, 2010.

[22] Y. Tian and J. M. Patel. Tale: A tool for approximate large graph matching. In *ICDE*, pages 963–972, 2008.

[23] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE Conference*, pages 405–416, 2009.

[24] P. T. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, 2012.

[25] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, pages 335–346, 2004.

[26] Z. Zeng, A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. Comparing stars: On approximating graph edit distance. *PVLDB*, 2(1):25–36, 2009.

[27] S. Zhang, M. Hu, and J. Yang. Treepi: A novel graph indexing method. In *ICDE*, pages 966–975, 2007.

[28] S. Zhang, S. Li, and J. Yang. Gaddi: distance index based subgraph matching in biological networks. In *EDBT*, pages 192–203, 2009.

[29] S. Zhang, J. Yang, and W. Jin. Sapper: Subgraph indexing and approximate matching in large graphs. *PVLDB*, 3(1):1185–1194, 2010.

[30] L. Zou, L. Chen, and M. T. Özsu. Distance-join: Pattern match query in a large graph database. *PVLDB*, 2(1):886–897, 2009.