Semantics-driven Approximate Query Answering on Graph Databases*
(Extended Abstract)

Federica Mandreoli[1], Riccardo Martoglia[1], Wilma Penzo[2], and Giorgio Villani[1]

[1] DII - University of Modena e Reggio Emilia, Italy
{federica.mandreoli, riccardo.martoglia, giorgio.villani}@unimo.it
[2] DEIS - University of Bologna, Italy
{wilma.penzo}@unibo.it

Abstract. Several database application areas need to deal with graph-modeled datasets. The main features of these datasets are the largeness and the heterogeneity of the data, which make it impractical to answer exact queries. In this paper we present our recent research efforts in modeling flexible query answering capabilities in this context. Flexibility is captured by approximations both on the labels and on the structure of graph-based queries, by guaranteeing *semantically meaningful* relaxations only. In order to cope with the excess of results, we adapt a well-known top-k retrieval algorithm to our context. The good effectiveness and efficiency of our proposal are proved by an extensive experimental evaluation on different real world datasets.

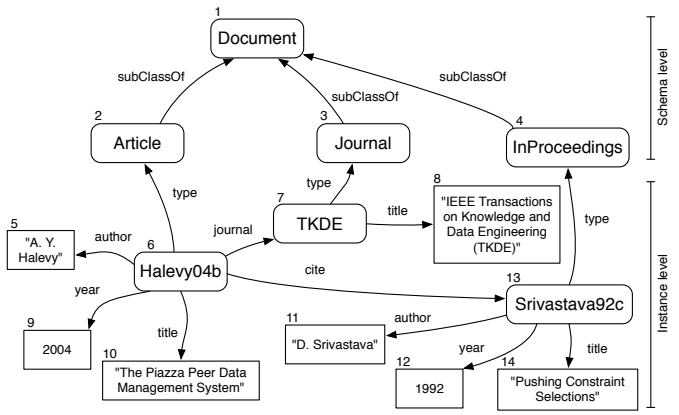
Keywords: Databases and the Semantic Web, Imprecise Information and Approximate Queries, Query Processing and Optimization, Data Models and Database Design

1 Introduction and Related Work

Graph-based data models have recently gained much popularity as powerful means for data representation in several database application areas [2, 5]. The main features of these datasets are the largeness and the heterogeneity of the data, which make it impractical to answer exact queries. This is mainly because of the lack of a complete knowledge of the vocabulary used, as well as of the information about how data is organized.

In order to overcome these problems, recent proposals [1, 4, 6] have adopted a keyword-based query model which has the merit of eliminating structures in the query, thus lightening the user from the burden of knowing the relationships occurring between the data. On the other hand, a keyword-based approach suffers from an inherently limited capability in the semantics it can express.

* The material presented in this paper has been published as [7] in the Proceedings of the 12th International Conference on Extending Database Technology (EDBT 2009). This work is partially supported by the Italian Council co-funded project NeP4B.



Query 1: The title of the papers authored by someone whose name sounds like 'Sivastava' published before 2005.
Query 2: The documents related in some way to the 'Piazza' paper published in the TKDE journal.

Fig. 1. Reference graph and two query samples

For instance, let us consider the graph-based data in Fig. 1, which represents a very small portion of the DBLP graph[3] in a RDF-like style, and let us look at Query 1. By following a simple keyword search query model [4, 6], and assuming for simplicity to disregard the 'Sivastava' misspelling and the condition on the year of publication, a user would translate Query 1 into the set of terms `Q1={title, paper, author, Srivastava}`. The main drawback of this kind of approach is that terms are allowed to occur anywhere in the data. This means that, for instance, papers having `Srivastava` as reviewer would appear in the result, even if they are obviously not relevant for the query.

From the above example it is evident the need of supporting semantic query capabilities which go beyond keyword-based search when searching for knowledge [5]. Essentially, the users should be enabled to include varying degrees of structure in their queries, so that they can better specify their own needs according to the partial knowledge of the schema they may have. Furthermore, it is of fundamental importance to get *meaningful* answers, i.e., answers made up of data related in a significant way.

These new demands pose new challenges as to the definition of appropriate and effective models for query specification and query answering, as well as of efficient algorithms and data structures to support their applicability.

Only few recent works have dealt with these issues so far [2, 5, 8]. [2] proposes a partial solution to the lack of expressiveness of the keyword-based approach by allowing the contextualization of value terms (e.g., `Srivastava`) into label terms (e.g., `author`). However, it does not allow the specification of relationships between terms. Thus, for instance, Query 2 in our reference example can

[3] Available at `http://sw.deri.org`

not be expressed. NAGA [5] and TALE [8], instead, adopt a graph-based query language which enables queries with the specification of links (i.e., relationships) between terms. Their answer model is then based on approximate graph matching. However, the approximation techniques like those adopted in [8] are often not adequate: they only consider data linkage, completely disregarding the semantics underlying the connections between the data. NAGA [5] presents a more powerful query language, but it does not delve into details as to the efficiency of the answering process. Finally, up to now most efforts have focused on the problem of approximating the structure of a query [8], often giving little attention to support vagueness on the vocabulary used. For this purpose, some works [2] adopt term expansion techniques. However, they do not explicitly deal with vague specifications on data links.

In this paper we address the problem of *approximate query answering* on graph-modeled data, namely finding data subgraphs which are *similar* to a query graph. We present a general data model which supports labeled directed/undirected data graphs with labeled/unlabeled edges (Sect. 2). We support approximate queries through: 1) node/edge label mismatches, and 2) structural relaxations. As for the latter point, a key contribution is the introduction of the notion of *Semantic Relatedness* (SR) relation for nodes in a data graph. The definition of SR makes our approach completely semantics-driven, since it admits *semantically meaningful* structural relaxations only (Sect. 3). We complete the work with a querying algorithm which efficiently generates the top-k answers according to a ranking model which takes into account approximations on both query nodes and edges (Sect. 4). Finally, we discuss the effectiveness and the efficiency of our proposal through experimental evaluation (Sect. 5).

2 Data Model

In this paper we present a flexible model for supporting approximate queries on graph-modeled data. Our purpose is to deal with a general data model which is not bound to any specific graph-based data model. Data is generically represented as a connected multigraph (i.e., a graph with parallel edges) with node and edge labels. In this way, we are able to cover most data models like the widely adopted standard RDF and interesting proposals recently introduced for various purposes (e.g. [2, 5, 6]).

A data graph essentially represents a portion of the real world through entities, values, and relationships between them [7]. As reference example we will use the graph shown in Fig. 1, which describes the relationship between two publications.[4] For simplicity, in Fig. 1 we show only a small portion of a potentially large and heterogeneous data graph which in most domains represents the collection of several different data sources. Thus, for instance, scientific papers are possibly described by concepts such as `Paper`, `Article`, `Publication`, `InProceedings`, and so on. It is worth noting that the support to parallel edges, which not all

[4] For ease of reference, nodes are univocally identified by the integer numbers i shown on the left upper corner and will be referenced as n_i.

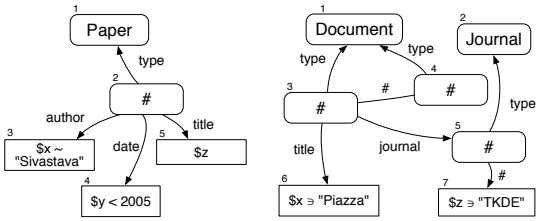


Fig. 2. Query 1 (left) and Query 2 (right)

models include (e.g. [8]), is instead fundamental in many contexts such as RDF and OWL based data as well as dataspaces, where the edge semantics depends not only on the involved nodes but also on the edge properties.

Given a data graph $\mathcal{G}$, we refer to a *path* as a sequence of consecutive edges not necessarily having the same direction.

3 Query Model

Our main aim is to go beyond the keyword-based approach without necessarily requiring to precisely use graph vocabulary and structure in query formulation. Rather, our query model allows users to specify query graphs exploiting whatever partial knowledge they may have. Queries can contain imprecise specifications both about the node and edge labels, and in the relationships among nodes.

In our framework, a *query* is expressed as a labeled multigraph connecting entity nodes and predicates on values. The query specification mechanism we propose allows for the formulation of graphs expressing as much topology and annotation as the user can, including none at all. To this end, users can annotate any node or edge with the wildcard "any label", '#', when they do not know the label at all. Indeed, the symbol '#' may be substituted by any of the labels. Moreover, users are allowed to specify general connections between nodes by means of undirected edges. Predicates on values have the form $var\langle op\rangle v$, where var is a variable, $\langle op\rangle$ is an operator, and v is a value. Possible operators are the relational ones (i.e., $=$, $<$, $\leq$, $>$, $\geq$) and the two operators $\ni$ (containment in a text) and $\sim$ (similarity between literals). Fig. 2 depicts the graphs of the two queries specified in Fig. 1. They both make use of the symbol '#' to denote the user's absence of knowledge about which entity nodes the specified terms are related to in the data (e.g., `Paper` and `author` in Query 1), and which relationships occur between such entity nodes (e.g., between the two document instances in Query 2). None of them finds an exact match on the reference graph because of node/edge label and structural mismatches: e.g. `Paper` and `date` in Query 1, the connection between n_1 and n_3 in Query 2.

Our query answering model relies on approximate subgraph matching in a two-fold fashion. The former kind of approximation we consider is node/edge label mismatch. The degree of mismatch between labels is quantified by means of a semantic distance function d_L that, for any pair of labels, returns a value ranging from exact match (0) to total mismatch (1). Obviously, for all labels l,

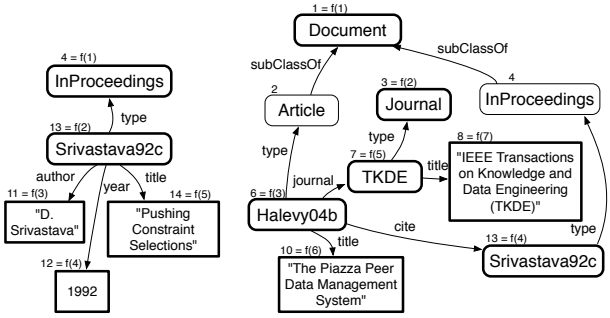


Fig. 3. Embeddings for Query 1 (left) and for Query 2 (right)

$d_L(\#, l) = 0$, and $d_L(l, l) = 0$. We let the users customize the label matching method that best suits the application needs. See [7] for a detailed discussion.

The latter kind of approximation we support is structural mismatch. A possible solution to support structural approximation is the purely topological approach which relaxes adjacency constraints by allowing node/edge insertions in the data graph. However, it should be noted that the fact that nodes are topologically connected does not necessarily imply that they are meaningfully related. For instance, notice that 2004 is not the year of publication of the paper `Srivastava92c` in Fig. 1 and thus the node pair (n_9, n_{13}) should not be considered when approximating Query 1.

This leads to the notion of *Semantic Relatedness (SR)* on a data graph $\mathcal{G}$ as a binary relation that only contains node pairs in $\mathcal{G}$ which are meaningfully related [7]. Each instance $e = (n, n') \in SR$ is assigned a label, a path $p(e)$ in $\mathcal{G}$ connecting n and n', and an approximation cost $c(e) \geq 0$. In other words, each instance $e \in SR$ is a "virtual" edge which $p(e)$ approximates in $\mathcal{G}$, and $c(e)$ expresses the cost to approximate e with $p(e)$. It follows that $E \subseteq SR$, where E is the set of edges in $\mathcal{G}$, and, since no structural approximation is required for each edge $e \in E$, for each of them the approximation cost is $c(e) = 0$.

Starting from SR, an SR-answer to a query graph q is an *approximate embedding* $\mathcal{E}^{SR} = (f, g)$ which defines a subgraph of $\mathcal{G}$ such that 1) each node n in q corresponds to a node $f(n)$ in $\mathcal{G}$, with f injective node-assignment function such that $d_L(\lambda(n), \lambda(f(n))) < 1$[5], and/or $f(n)$ satisfies a query condition; 2) each edge $e = (n, n')$ in q corresponds to a pair $g(e) = (f(n), f(n'))$ in SR, with g injective edge-assignment function. $g(e)$ possibly approximates e's label and possibly relaxes e according to SR. Thus, an approximate embedding $\mathcal{E}^{SR}$ combines both label and structural matchings. Fig. 3 shows one plausible approximate embedding for each of the two query samples.[6] For ease of presentation, the subgraph shown for Query 1 only contains label approximations, whereas the one shown for Query 2 only deals with structural approximations.

[5] The function $\lambda(\cdot)$ returns the label of a node.

[6] The data nodes in the range of f are depicted in bold line and the query node image is shown on the left upper corner of each data node rectangle. For instance, $4 = f(1)$ means that data node n_4 is assigned to query node n_1.

In a very frequent scenario where a large number of approximate embeddings are returned for a given query, we are concerned with finding the *top-ranked answers*. To this end, we measure answer goodness by applying a scoring function $\mathcal{S}$ to each embedding $\mathcal{E}^{SR}$. $\mathcal{S}$ combines the approximations occurring at both data nodes and edges, including the approximations on query conditions, and it returns a score in $[0, 1]$ such that the higher is $\mathcal{S}(\mathcal{E}^{SR})$ the higher is the overall approximation required for the matching. For more details on $\mathcal{S}$ see [7].

4 Efficient Support to Query Answering

The most expensive operation during query processing is to check whether two data nodes are semantically related or not under the label constraints specified in the query. Our aim is to reduce space and time complexity of search by performing some offline operations. To this end, we decided to create SR indices. In particular, we propose to index SR elements according to node and edge labels. Elements can be accessed either sequentially using cursors or directly exploiting further indices which consider not only labels, but also node ids.

We exploit such indices in a querying algorithm which efficiently generates the top-k answers in an order that is correlated with the ranking measure. The algorithm builds on the principles of the Threshold Algorithm (TA) proposed in [3]. For lack of space, we only sketch the querying algorithm, and we refer the reader to [7] for a more detailed description. Initially, the algorithm searches for the query node labels (possibly approximate) in the data, and associates each query edge to a list of SR elements sorted according to the approximation cost w.r.t. the query specification. Then, the algorithm performs a sorted access in parallel to each of the lists. As an element is seen in some list (object in TA), it does random access to the other lists to compute answers. For each answer, the algorithm computes its score and, if it is one of the k lowest, it remembers it. After each computation step, the algorithm computes the score *lBound* of the set of the next node pairs under sorted access to the lists, and stops the process whenever at least k answers have been seen whose grade is smaller than *lBound*. SR indices are exploited during lists and answer construction.

To give an intuition of its working, let us consider Query 1 in Fig. 2. First, the algorithm searches for labels 'Paper', 'Sivastava', aso, e.g., retrieving 'Article' and 'InProceedings' for 'Paper'. Then, it deals with query edges by associating, for instance, the edge ('#'-type-'Paper') to the list made of ('Halevy04b'-type-'Article') and ('Srivastava92c'-type-'InProceedings') which involve the terms 'Article' and 'InProceedings' retrieved for 'Paper'. Finally, it tries to build the complete answers by accessing the other lists, each one associated to one of the remaining edges in Query 1.

The algorithm operates in a more challenging scenario than other TA-derived algorithms presented in the literature (such as [4]) and thus introduces several enhancements and modifications w.r.t. them:

- each element in one list conceptually joins with more than one element in each of the other lists;

				answers	equiv queries		precision
Query	#n	#e	#any	#exp	#exact	#web	P
DBLP-S Dataset							
Q1	3	2	1	12	1	3	1
Q2	3	2	1	36	4	2	1
Q3	5	4	3	10	25	n/a	1
Q4	6	6	4	148	64	41	1

Table 1. Effectiveness results - DBLP-S

- the fact that an element has been already seen during answer construction does not mean that all the solutions involving it have been generated;
- as to list selection, differently from [4] where only round-robin accesses are considered, our algorithm includes additional selection strategies whose goal is to try to build better ranked answers as soon as possible. For instance, we can choose the list minimizing the approximation (next-best strategy). Moreover, as our lists are not equally sized, we can start building answers by prioritizing the list whose size is the smallest one (max-sel strategy).

For a complete description of the algorithm we refer to [7].

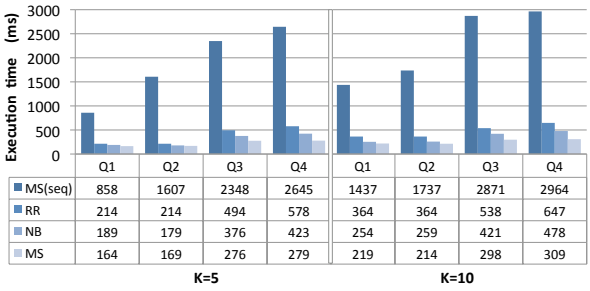
5 Experimental Evaluation

We performed an extensive experimental evaluation on several RDF-based real world datasets. In particular, we relied on a RDF-like instantiation of SR which, due to lack of space, we can not present. The interested reader is referred to [7].

We selected two collections extracted from the RDF version of the DBLP scientific bibliography data, DBLP-S and DBLP-L. We considered a set of significant queries, named Q1-Q4. Specifically, Q1 asks for "Articles of year 1997", Q2 for "Papers (generically) of year 1997", Q3 for "Titles of papers of date 1990 and semantically connected (# edge) to STACS conference", and Q4 for "Titles of documents created by a person who has also created a document in 1994". Queries are designed to be increasingly more complex, both in terms of number of nodes and unspecified labels.

The algorithm and data structures are implemented in Java 1.6, exploit MySQL 5.0 and Oracle BerkeleyDB 3.2 JE storage engines and indices. All the experiments are executed on an Intel Core2 Duo 2.4Ghz OSX workstation, equipped with 2GB RAM.

Tab. 1 presents a selection of the effectiveness results we obtained, along with an overview of the numeric features of the employed queries (number of nodes, number of edges, number of "any labels" '#', number of expected answers). Queries typically require both structural and semantic approximations to be correctly solved. The table shows different measures and comparisons, from left to right: the number of queries that would be necessary to obtain all the expected results from an exact matching approach, the number of queries that would have to be submitted to DBLP web search portals, and the precision (i.e. percentage of relevant retrieved answers w.r.t. the retrieved ones) of our approach. Results are: Q1 does not require approximations, thus both the exact approach and ours return the correct answer, with precision 1. However, notice that the number



	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
MS(seq)	858	1607	2348	2645	1437	1737	2871	2964
RR	214	214	494	578	364	364	538	647
NB	189	179	376	423	254	259	421	478
MS	164	169	276	279	219	214	298	309

Fig. 4. Query Execution time for DBLP-L

of exact queries grows when we introduce approximations. Tab. 1 also shows that retrieving all the relevant answers through the web DBLP search engine would require a significant work from the user. Further, note that Q3 cannot be performed through the web search portal.

As to efficiency, we considered query execution time under our list selection strategies. In particular, Fig. 4 compares query execution time of Q1-Q4 executed according the round robin (RR), next-best (NB) and max-sel (MS) strategies exploiting indexed access to lists w.r.t. the max-sel strategy with a sequential access (MS(seq)), for both K=5 and K=10. Execution time reflects the query complexity. Further, sequential versions of access to elements proved significantly slower than their index-based counterparts.

6 Conclusions

We presented a general model for supporting approximate queries on graph-modeled data. We introduced the notion of Semantic Relatedness to allow for *semantically meaningful* relaxations only. Further, we provided an efficient support to query answering, properly tested by extensive experiments.

References

1. G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *ICDE*, 2002.
2. X. Dong and A. Halevy. Indexing dataspaces. In *SIGMOD*, 2007.
3. R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. In *PODS*, 2001.
4. H. He, H. Wang, J. Yang, and P. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, 2007.
5. G. Kasneci, F. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and Ranking Knowledge. In *ICDE*, 2007.
6. G. Li, B. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data. In *SIGMOD*, 2008.
7. F. Mandreoli, R. Martoglia, W. Penzo, and G. Villani. Flexible Query Answering on Graph-modeled Data. In *EDBT*, 2009.
8. Y. Tian and J. Patel. TALE: A Tool for Approximate Large Graph Matching. In *ICDE*, 2008.