International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552 Volume- 8, Issue- 6, November 2020
https://doi.org/10.21276/ijircst.2020.8.6.5
www.ijircst.org

On the Feasibility of Handling Uncertainty in SPARQL Queries in the Case of Sparse Graphs

Theodore Andronikos

ABSTRACT- The purpose of this paper is to explore the possibility of evaluating SPARQL queries containing probabilities using linear algebraic methods and techniques. This approach has many important advantages, such as simplicity, succinctness, elegance and greater familiarity to a wide base of practitioners. On the other hand, there are questions regarding its efficiency in view of the huge datasets of the current era. In this paper, we advocate that in the case of sparse RDF graphs we can resort to sparse matrices for computing complicated probabilistic queries. It is demonstrated that via probabilistic sparse matrices one can evaluate specific types of queries involving transitive predicates, which are of great practical importance. The algorithm and data structures that are currently available for handling sparse matrices promise improved performance in pragmatic situations, which constitutes this line of approach particularly promising.

KEYWORDS- RDF graph, SPARQL query, probabilistic SPARQL query, sparse matrices, probabilistic matrices.

I. INTRODUCTION

Today most aspects of everyday life activities have something to do with the Web. It would not be an exaggeration to claim that research about the Web and the related state of the art technologies dominates the field of Computer Science. The enormous and often chaotic information distributed all over the internet must be stored and processed as efficiently as possible. The storage part is taken care of by the Linked Open Data [17] and the Resource Description Framework (RDF) [23]. Retrieving the information contained in the, usually enormous, datasets is typically achieved by the specialized query language SPARQL [28].

From a technical viewpoint, RDF datasets can be modeled as directed graphs. In the present era, the size of these graphs is typically huge. This fact presents many algorithmic challenges and, at the same time, is the source of innovative research. Simplicity is perhaps the best, if not the only, way to tackle gigantic datasets. This is the main purpose behind the syntax of triples, which is adopted by the RDF standard: (s,p,o), where s, p and o stand for subject, predicate and object, respectively. The meaning of the triples is that the subjects are connected to the objects via the relations expressed by the predicates.

On should be aware however that in many occasions the facts stored in the dataset may be inaccurate, or even completely false. In a grey world, it is not always possible to express absolute truth in triples. Simply put, the truth of the RDF triples maybe uncertain, or, more accurately, certain up to a degree of probability. Therefore, it has been suggested by many researchers that it could prove useful to associate to each triple a numerical value in the interval $[0,1]$. This numerical value may be called in various contexts probability, or degree of confidence, or extent of certainty. Irrespective of the term used, the idea is the same: it is not absolutely certain that the triple in question is true. There is some ambivalence, the measure of which is reflected by the numerical value. Such datasets are rather common and a characteristic example is that of biological data [13]. The present paper examines under what circumstances it may be possible to make inferences in the presence of incomplete or ambivalent information.

SPARQL is the most prominent query language for RDF datasets. The feature of SPARQL that will be of most use in this study is its ability to handle path queries. Using path queries on can capture paths in the underlying RDF graph. For this endeavor to be meaningful, it is necessary for the RDF graph to contain at least one transitive predicate. An RDF predicate T is transitive, if it satisfies the transitive property, that is (u,T,w) can be inferred from (u,T,v) and (v,T,w). Traversing consecutive edges labeled by T amounts to traversing a path, which is called for brevity T-path, in the RDF graph.

A. Overview of previous work

Let us now briefly mention a few important papers that bear some relevance to this work.

The authors in [30] considered path queries formed by context-free grammars. It is a well-known theoretical fact that context-free grammars are more expressive than regular expressions. Motivated by this fact, the authors designed the query language cfSPARQL. This language is a strictly more powerful extension of SPARQL and, as such,

Manuscript Received November 20, 2020

Theodore Andronikos, Associate Professor, Department of Informatics, Ionian University, 7 Tsirigoti Square, Corfu, Greece, (e-mail: andronikos@ionio.gr)

it provides mechanisms to write queries that cannot be formulated in standard SPARQL. An attempt to classify SPARQL queries from an algebraic viewpoint was made in [26]. This led to the characterization of queries in terms of equivalence classes. In the same vein, [1] proposed and examined another concept of equivalence, which produced different families of similar queries. The researchers in [25] made insightful connections between queries and automata, that demonstrated the feasibility of formulating queries via automata-theoretic means. Another interesting association of queries on Linked Data and Path Queries to Büchi automata and ω-automata was established in the works of [9] and [10]. Similarly, the possibility of query evaluation via automata was explored in [29]. The study of the feasibility of inclusion of extra temporal information and the resulting enhanced expressive power was undertaken in [31], [4] and [5] using different approaches.

Although initially met with skepticism, particularly in view of the question of practical efficiency, the idea of adding probabilities to RDF triples has by now attained substantial recognition. This would lead to a more accurate reflection of the situation by the explicit facts contained in the dataset, and would also allow the expression and evaluation of probabilistic queries. A great deal of research has been conducted towards this direction, from theoretical guidelines in [24] to more practical considerations. The researchers in [13] pioneered an extension of SPARQL with the aim of handling queries on probabilistic RDF graphs. A study of probabilistic ontologies together with techniques for answering related queries was presented in [27]. Uncertainty in path queries was discussed in [12], while [16] undertook the task of augmenting RDF triples with probabilities. A good source for further reading is also reference [14].

An important step was also the introduction of pSPARQL in [8] and its subsequent extension in [7]. Standard SPARQL is a proper subset of pSPARQL, which is geared towards formulation of probabilistic queries. The idea that probabilistic queries can be handled via the use of suitable complex matrices is proposed and explained in [2]. There it was remarked that often unconventional techniques, inspired by the field of quantum computing, can provide an edge over classical methods, as was elaborated in references [18], 19], and [20]. It has also been demonstrated that probabilistic automata can be utilized in answering probabilistic queries in [3].

The contribution of this paper is the proposal of sparse matrices as a useful and efficient method for computing complicated probabilistic queries on sparse RDF graphs. It is shown here that, in principle, the use of probabilistic sparse matrices can make practically possible the fast computation of certain important categories of queries involving transitive predicates. This method can further be augmented with ad hoc criteria, which make assessments in terms of the current probability value attributed to a triple. Successive multiplication of probabilistic matrices tends to rapidly lower the probability values contained in the resulting matrices or vectors. Probabilities close to zero suggest improbability rather than probability, thus, it may be expedient to regard the corresponding inferred triple as false or nonexistent. Irrespective of the adoption or not of such ad hoc heuristics, the algorithm and data structures currently employed for handling sparse matrices can guarantee very satisfactory performance in real life situations.

II. BACKGROUND

This section contains the definitions and the notation used in the rest of the paper. We assume that we are dealing with sparse probabilistic RDF graphs, that is, the graphs are sparse and to each triple (s,p,o) corresponds a real number in the interval $[0,1]$, which quantifies the degree of certainty of the particular fact.

Definition 1. Suppose we are given the column vector $\boldsymbol{u} \in \mathbb{R}^n$, where $\boldsymbol{u}^T = [u_1 \quad u_2 \quad \dots \quad u_n]$ ($\boldsymbol{u}^T$ is the transpose of $\boldsymbol{u}$). We say that $\boldsymbol{u}$ is *stochastic*, if $\forall i \; u_i \geq 0$ and $\sum_{i=1}^{n} u_i = 1$, and $\boldsymbol{u}$ is *substochastic*, if $\forall i \; u_i \geq 0$ and $\sum_{i=1}^{n} u_i \leq 1$. Similarly, a square matrix $\boldsymbol{T} \in \mathbb{R}^{n \times n}$ is *stochastic*, if all its columns are stochastic and substochastic if all its columns are *substochastic*.

Definition 2. Given a sparse RDF graph, let T be a transitive predicate appearing in the graph. Let us further assume that the vertices that are incident to arcs labeled by T are $1, 2, \dots, n$. We construct the $n \times n$ matrix $\boldsymbol{T}$ whose elements $t_{ij}, 1 \leq i,j \leq n$, are defined as follows:

$$t_{ij} = \begin{cases} 0 & if\ (i,T,j)\ does\ not\ exist, \\ p_{ij} & if\ (i,T,j)\ exists\ and\ is\ assigned\ probability\ p \end{cases}.$$

Definition 3. We may define a real number $c \in [0,1]$, which we call *threshold*, such that whenever the probability p corresponding to an inferred triple is below the threshold, this triple is considered improbable and is dismissed. Practically, this means that in the corresponding matrix entry p becomes 0.

Sparse RDF graphs are common. Some authors suggest that in real life scenarios most large matrices are sparse [33]. Hence, it is undeniably useful to consider the case of sparse RDF graphs. Today there exist highly efficient data structures for the representation of sparse matrices. It is worth pointing out that the dominant operation may affect the choice of the underlying data structure. Taking into account that the sparse RDF graph is represented by a sparse matrix and the nodes of this graph are also conveniently represented by (column) vectors, suggests that the two main operations are matrix vector multiplication and matrix matrix multiplication. For these operations, using appropriate storage schemes, such as Compressed Sparse Row, or Compressed Sparse Column, can lead to substantial performance gains compared to the standard methods of numerical analysis (see [22] and [11]). For instance, in the case of matrix vector multiplication, it is possible to achieve time linear in the size of the nonzero elements of the sparse matrix. For more details, the interested reader may consult [6], [21], [32] and [34].

III. MAIN RESULTS

Simple standard SPARQL queries can retrieve information that is explicitly stored in the dataset. Since SPARQL 1.1 [28], it has become possible to make use of its enhanced navigational capacity using transitive predicates.

In this manner, it is possible to retrieve implicitly stored information from the dataset, as long as this information can be acquired by traversing a path with edges labeled by one or more transitive predicates. In a probabilistic setting, in addition to the path characteristics, we must compute the likelihood of the information being true. Establishing that node v is related to node u by a T-path, i.e., a path labeled by the transitive predicate T is not the end of the story. The degree of certainty of this fact must be calculated, so that we may assess if the inferred fact is very probable or highly unlikely.

Definition 4. We say that queries of the following form are *simple one-source* queries.

```
SELECT ?d
WHERE {
    s T+ ?d .
}
```

Simple one-source query.

This type of query is formulated in order to retrieve all nodes that are reachable via a T-path from a specific source node designated by s. The aforementioned path has length at least 1 and its edges are exclusively labeled by the transitive predicate T.

Queries of the next form are *simple one-destination* queries.

```
SELECT ?s
WHERE {
    ?s T+ d .
}
```

Simple one-destination query.

A query such as the above will return those source nodes from which emanate T-paths (of length at least 1) that end up at the destination node called d.

Similarly, we may designate *simple many sources - many destinations* queries.

```
SELECT ?s ?d
WHERE {
    ?s T+ v .
    v T+ ?d .
}
```

Simple many sources - many destinations query.

This kind of query fixes a specific node, which is called node v, and collects pairs of initial and the terminal nodes that define paths passing through node v. Specifically, the first coordinate of the pair is the initial source node from which the path begins and the second coordinate of the pair is the terminal node of the path. The path itself must consist of two sub-paths: the first from the initial node to node v, and the second from node v to the terminal node. Both paths are T-paths of length at least 1.

Queries involving more than one transitive predicates are called *composite* queries. Using two or more transitive predicates we may define composite *one-source* queries with the following general form.

```
SELECT ?d
WHERE {
    s T1+ ?x .
    ?x T2+ ?d .
}
```

Composite one-source query.

Queries such as the above are *composite one-source* queries. They can be generalized even further by the inclusion of more than two transitive predicates, i.e., T_3, T_4, and so on. This query will retrieve all nodes that are reachable via a composite path from a specific source node denoted by s. The path is characterized as composite in the sense that it is the concatenation of two different paths, each of length at least 1. The first path is a T_1-path, meaning that the arcs are labeled by predicate T_1 and the second path is a T_2-path, i.e., the arcs are labeled by predicate T_2.

Analogously, we have composite *one-destination* and composite *many sources - many destinations* queries.

```
SELECT ?s
WHERE {
    ?s T1+ ?x .
    ?x T2+ d .
}
```

Composite one-destination query.

The previous query will return all nodes from which a composite path, ending at a specific destination node denoted by d, originates. In this case too the path is considered to be composite because it is the concatenation of two different paths, each of length at least 1. The first path is a T_1-path, meaning that the arcs are labeled by predicate T_1 and the second path is a T_2-path, i.e., the arcs are labeled by predicate T_2.

```
SELECT ?s ?d
WHERE {
    ?s T1+ v .
    v T2+ ?d .
}
```

Composite many sources - many destinations query.

The above query has arguably the most complicated form. It will retrieve pairs of nodes, the first node playing the role of source, and the second node playing the role of destination. Every pair of source and destination nodes defines a composite path passing through a designated node

v. The path in question is composite, as it consists of two smaller paths (of length at least 1). The first is a T_1-path, and the second is a T_2-path.

Common characteristics to all the previous classes of queries is the specification of a path or paths via the operator + combined with the transitive predicate or predicates (see [15]). The number of types of paths depends on the number of different transitive predicates that appear in the query. In the case of simple queries we encounter just one type of path, whereas in the case of composite queries we encounter at least two types of paths. In all cases that paths have length *at least* 1, as this is a consequence of the application of + operator.

The above classes of queries are of obvious practical importance, as they enable the synthesis and retrieval of path properties from the underlying dataset. Moreover, they can serve as a basis for the definition of more complicated queries, in many different ways, i.e., containing more predicates or resulting from the conjunction of previously defined queries.

Let us now recall that we have assumed an (implicit) ordering $1, 2, \ldots, n$ of the vertices incident to each transitive predicate T. This will be useful in the formulation of the vectors and matrices that will be required during the computation process.

Definition 5. In the present linear algebraic setting, the vertices can be represented by unit n dimensional column vectors using the following simple mapping:

$$k \mapsto \boldsymbol{q}_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \mapsto (k^{th}\ coordinate), 1 \le k \le n. \quad (1)$$

Definition 6. Given a column vector $\boldsymbol{q}$, the set of indices of the nonzero elements of $\boldsymbol{q}$ are denoted by $NZ(\boldsymbol{q})$.

The construction outlined in Definition 2 will produce the $n \times n$ matrix $\boldsymbol{T}$. Ideally, this matrix should be stochastic. It may, however, be more realistic to expect $\boldsymbol{T}$ to be substochastic. We shall argue that in the case of sparse matrices it is practically feasible to evaluate queries of the classes summarized in Definition 4, which constitute an important and useful subset of SPARQL queries. Computing these queries involves probabilistic $\boldsymbol{T}$ matrices, one for each transitive predicate appearing in the query. In theory at least, it will require the computation of powers of the matrix $\boldsymbol{T}$, i.e., $\boldsymbol{T}^m$ for some $m \ge 1$. The following Proposition 1, whose proof is trivial and is omitted, presents a well-known fact regarding such matrices.

Proposition 1. Consider the $n \times n$ matrix $\boldsymbol{T}$, as in Definition 2. Let $\boldsymbol{T}^m$ be the m^{th} power of $\boldsymbol{T}$, $m \ge 1$, and let $t_{ij}^m, 1 \le i,j \le n$ be its elements. The elements t_{ij}^m provide information about paths of length m from node i to node j in the original RDF graph. Specifically, if $t_{ij}^m = 0$, then there is no path of length m from i to j. Otherwise, if $t_{ij}^m = p$, for some real number $p \in [0,1]$, then there exists a path of length m from i to j with probability p.

□

In the above Proposition 1, when we refer to paths of length m, we mean paths of length m where all arcs are labeled by the transitive predicate T. Sometimes to emphasize this fact we call them T-paths. Perhaps it is worth pointing out that the existence or not of a path of length m neither implies, nor precludes the existence of a path of length $m+1$. Another issue that may come into play is the application or not of the threshold. The probability assigned to the path is the measure of the certainty about the existence of the path. If the probability is very low then the path probably does not exist. According to the precision constraints at hand, it might be advantageous to view probabilities lower than a threshold as zero, thus ignoring the corresponding matrix element. This will increase the sparsity of the matrix and make even more efficient its subsequent use. The existence of a T-path from i to j with assigned probability p, is equivalent to the inference of the triple (i, T, j) with degree of certainty p.

A. Computing simple one-source queries

Simple one-source queries are easy to compute in principle. Let $\boldsymbol{q}_s = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ be the vector representation of source vertex s.

Invoking Proposition 1 and using a simple induction on m, it is straightforward to show the following Proposition.

Proposition 2. Given a simple one-source query involving the transitive predicate T and the source node s, the *destinations at distance* m, $m \ge 1$, from node s are given by the nonzero entries of vector $\boldsymbol{q}_{d,m}$, where:

$$\boldsymbol{q}_{d,m} = \boldsymbol{T}^m \boldsymbol{q}_s . \quad (2)$$

The set of destination nodes, at any distance from s, is given by $D(s)$, where:

$$D(s) = \bigcup_{m=1}^{n} NZ(\boldsymbol{q}_{d,m}) . \quad (3)$$

We note that the above set $D(s)$ also contains the nodes adjacent to s. □

In the case of a sparse matrix $\boldsymbol{T}$, it is more efficient to compute this vector by successive matrix vector multiplications, i.e., $\boldsymbol{T}\boldsymbol{q}_s$, $\boldsymbol{T}^2\boldsymbol{q}_s$, …, $\boldsymbol{T}^{m-1}\boldsymbol{q}_s$ and $\boldsymbol{T}^m\boldsymbol{q}_s$. First, the multiplication of a vector by a sparse matrix requires time linear in the number of nonzero elements of the matrix. Second, it may happen that some entries in a vector appearing in this sequence of computations are below the threshold. In this case, they can be simply regarded as zeros, which will facilitate even more the remaining calculations.

B. Computing simple one-destination queries

Symmetrically, one can easily compute simple one-destination queries. If $\boldsymbol{q}_d$ is the vector representation of destination vertex d, then by Proposition 1 and a simple induction on m, it can be proved that:

Proposition 3. Given a simple one-destination query involving the transitive predicate T and the destination node

International Journal of Innovative Research in Computer Science & Technology (IJIRCST)
ISSN: 2347-5552 Volume- 8, Issue- 6, November 2020
https://doi.org/10.21276/ijircst.2020.8.6.5
www.ijircst.org

d, the *sources at distance* $m, m \geq 1$, from node d are given by the nonzero entries of vector $\boldsymbol{q}_{d,m}$, where:

$$\boldsymbol{q}_{s,m} = (\boldsymbol{T}^{\boldsymbol{T}})^{\boldsymbol{m}} \boldsymbol{q}_d \,. \qquad (4)$$

The set of source nodes, at any distance from d, is given by $D(d)$, where:

$$S(d) = \bigcup_{m=1}^{n} NZ(\boldsymbol{q}_{s,m}) \,. \qquad (5)$$

We note again that the set $D(d)$ contains the nodes adjacent to d. □

It is important here to emphasize that $\boldsymbol{T}^{\boldsymbol{T}}$, which is the transpose of matrix $\boldsymbol{T}$, must be used. The previous remarks regarding the efficiency of the matrix vector multiplication for sparse matrices, also apply in this case.

C. Computing simple many sources - many destinations queries

Simple many sources - many destinations queries return ordered pairs (i, j) of nodes such that there is a path from i to a specific node v and a path from v to j. The paths have length at least 1. If $\boldsymbol{q}_v$ is the vector representation of vertex v, then by combining together Propositions 2 and 3, we derive the next proposition 4.

Proposition 4. Given a simple many sources - many destinations query involving the transitive predicate T and the intermediate node v, the source nodes at distance $m_1, m_1 \geq 1$, from node v are given by the nonzero entries of vector $\boldsymbol{q}_{s,m_1} = (\boldsymbol{T}^{\boldsymbol{T}})^{\boldsymbol{m_1}} \boldsymbol{q}_v$, and the destination nodes at distance $m_2, m_2 \geq 1$, from node v are given by the nonzero entries of vector $\boldsymbol{q}_{d,m_2} = \boldsymbol{T}^{m_2} \boldsymbol{q}_v$. Hence, the ordered pairs (i, j) of nodes such that there is a path of length at least 1 from i to v and a path of length at least 1 from v to j, is given by:

$$i \in S(v) \textbf{ and } j \in D(v) \,. \qquad (6)$$

This set of pairs contains all the pairs of nodes that are adjacent v □

D. Computing composite one-source queries

When answering composite queries, the major difference compared to the previous cases of simple queries is the presence of two or more transitive predicates. This, in turn, causes the inclusion of two or more (substochastic) $n \times n$ matrices $\boldsymbol{T}_1, \boldsymbol{T}_2, \ldots$.

In the case of composite one-source queries, as the one shown in Definition 4, we can state the following Proposition 5, which is a generalization of Proposition 2.

Proposition 5. Given a composite one-source query involving the transitive predicates T_1, T_2, and the source node s, the nodes reachable from s via a T_1-path of length $m_1, m_1 \geq 1$, and a T_2-path of length $m_2, m_2 \geq 1$, are given by the nonzero entries of vector $\boldsymbol{q}_{d,m_1,m_2}$, where:

$$\boldsymbol{q}_{d,m_1,m_2} = \boldsymbol{T}_2^{m_2} \boldsymbol{T}_1^{m_1} \boldsymbol{q}_s \,. \qquad (7)$$

The set of destination nodes through a T_1-path and a T_2-path, at any distance from s, is given by $CD(s)$, where:

$$CD(s) = \bigcup_{m_1+m_2=2}^{n} NZ(\boldsymbol{q}_{d,m_1,m_2}) \,. \qquad (8)$$

We note that the above set $CD(s)$ also contains the nodes adjacent to s. □

E. Computing composite one-destination queries

Composite one-destination queries, the general form of which is shown in Definition 4, involve at least two transitive predicates. Consequently, their computation must make use of an equal number of $n \times n$ matrices, one for each predicate. The details of their computation is given in the next Proposition 6, which can also be viewed as a generalization of Proposition 3.

Proposition 6. Given a composite one-destination query involving the transitive predicates T_1, T_2, and the destination node d, the source nodes from which d is reachable via a T_1-path of length $m_1, m_1 \geq 1$, and a T_2-path of length $m_2, m_2 \geq 1$, are given by the nonzero entries of vector $\boldsymbol{q}_{d,m_1,m_2}$, where:

$$\boldsymbol{q}_{s,m_1,m_2} = (\boldsymbol{T}_1^{T})^{m_1} (\boldsymbol{T}_2^{T})^{m_2} \boldsymbol{q}_d \,. \qquad (9)$$

The set of source nodes from which originates a T_1-path followed by a T_2-path that eventually terminates at node d, is given by $CS(s)$, where:

$$CS(d) = \bigcup_{m_1+m_2=2}^{n} NZ(\boldsymbol{q}_{s,m_1,m_2}) \,. \qquad (10)$$

We note that the above set $CD(d)$ also contains the nodes adjacent to d. □

F. Computing composite many sources - many destinations queries

The composite many sources - many destinations case can be tackled in a similar way as the previous composite cases. Proposition 7 that is stated next contains the details and generalizes Proposition 4.

Proposition 7. Suppose we are given a composite many sources - many destinations query involving the transitive predicates T_1, T_2, and the intermediate node v. The nodes from which a T_1-path of length $m_1, m_1 \geq 1$, to v emanates are given by the entries of $\boldsymbol{q}_{s,m_1} = (\boldsymbol{T}_1^{T})^{m_1} \boldsymbol{q}_v$. Symmetrically, the nodes to which a T_2-path of length $m_2, m_2 \geq 1$, from v terminates are given by the entries of $\boldsymbol{q}_{d,m_2} = \boldsymbol{T}_2^{m_2} \boldsymbol{q}_v$. The required ordered pairs (i, j) of nodes such that a path of length at least 1 originates from i and ends at v and a path of length at least 1 starts from v and ends at j, are given by:

$$i \in CS(v) \textbf{ and } j \in CD(v) \,. \qquad (11)$$

The above set of pairs will also return all the pairs of adjacent nodes to v □

Answering queries that are more complex, involving more than two predicates, can be accomplished by straightforward generalizations of the above Propositions 5 - 7.

IV. CONCLUSION

In this paper, we have investigated the evaluation of certain practically important classes of probabilistic queries involving transitive predicates and paths. The ability to retrieve enhanced navigational information is a useful feature that offers concrete advantages to the practitioner. Moreover, in many environments where uncertainty is inherently present, the capability to reason about probabilities, particular when assessing inferred facts is indispensable. For these types of queries, we have shown how they can be computed using linear algebraic methods, and, specifically, appropriate matrix vector multiplications. When the matrices involved are dense, this approach is probably not very promising. Things are radically different when the matrices involved, that is the matrices corresponding to the transitive predicates, are sparse. Then this method becomes efficient. Taking advantage of state of the art data structures and techniques for sparse matrices, e.g., Compressed Sparse Row or Compressed Sparse Column, it is possible the perform the multiplication of the sparse matrix with the vector in time that is linear in the number of the nonzero elements of the matrix. The situation can be further improved by employing a threshold. The main idea behind the introduction of a threshold is very simple: any probability value below the threshold signifies some that very improbable. Therefore, by systematically comparing the probabilities that arise during the computational process with the threshold, and taking as zero those that are below the threshold, we reduce the overall number of the nonzero elements, which speads up the performance of this method.

REFERENCES

[1] Andronikos, T., "Classification of SPARQL queries into equivalence classes of relevant queries", International Journal of Advanced Research in Computer Science, December 2017, Volume 8, No. 9, pages 152-159.

[2] Andronikos, T., Complex Matrices for the Approximate Evaluation of Probabilistic Queries, International Journal of Engineering Research and Applications (IJERA), Vol. 10, Issue 11, (Series-I) November 2020, pp. 23-30.

[3] Andronikos T., Singh A., Giannakis K., Sioutas S., Computing probabilistic queries in the presence of uncertainty via probabilistic automata, Algorithmic Aspects of Cloud Computing, Third International Workshop, ALGOCLOUD 2017, Vienna, Austria, 5 September, 2017, Revised Selected Papers. Springer Theoretical Computer Science and General Issues, Volume 10739, pp. 106-122, 2018.

[4] Motik, B., Representing and querying validity time in RDF and OWL: A logic-based approach, Journal of Web Semantics, Elsevier BV, 2012, 12-13, 3-21.

[5] Andronikos T., Stefanidakis M., Papadakis I., Adding Temporal Dimension to Ontologies via OWL Reification, Proceedings of the 13th Panhellenic Conference on Informatics - PCI 2009 Conference, 10-12 September, Corfu, Greece, IEEE Computer Society, pp. 19-22, 2009.

[6] López, C. P., MATLAB Linear Algebra, Apress, 2014.

[7] Fang, H. pSPARQL: A Querying Language for Probabilistic RDF Data Complexity, Hindawi Limited, 2019, 1-7.

[8] H. Fang and X. Zhang, "pSPARQL: a querying language for probabilistic RDF (extended abstract)," in Proceedings of the ISWC'16, Posters, 2016.

[9] Giannakis K., Andronikos T., Querying Linked Data and Büchi Automata, IEEE Proceedings of the 9th International Workshop on Semantic and Social Media Adaptation and Personalization (SMAP), 6-7 November, Corfu, Greece, pp. 110 - 114, 2014.

[10] Giannakis K., Theocharopoulou G., Papalitsas C., Andronikos T., Vlamos P., Associating ω-automata to Path Queries on Webs of Linked Data, Engineering Applications of Artificial Intelligence, Elsevier, Volume 51, May 2016, pages 115-123.

[11] Lay, D., Linear algebra and its applications, Pearson, 2016.

[12] Hua, M., Pei, J.: Probabilistic path queries in road networks: traffic uncertainty aware path selection. In: Proceedings of the 13th International Conference on Extending Database Technology, pp. 347–358, ACM, 2010.

[13] Huang, H., Liu, C.: Query evaluation on probabilistic RDF databases. In: International Conference on Web Information Systems Engineering, pp. 307–320, Springer, 2009.

[14] Szeto C., Hung E., Y. Deng, SPARQL query answering with RDFS reasoning on correlated probabilistic data, in Proceedings of the WAIM'11, pp. 56–67, 2011.

[15] Zhang X., J. Van den Bussche, On the primitivity of operators in SPARQL, Information Processing Letters, vol. 114, no. 9, pp. 480–485, 2014.

[16] Lian, X., Chen, L., Wang, G.: Quality-aware subgraph matching over inconsistent probabilistic graph databases. IEEE Transactions on Knowledge and Data Engineering 28(6), 1560–1574, 2016.

[17] LOD Project, 2014. Linking Open (LOD) Data Project, http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData.

[18] Papalitsas C., Andronikos T., "Unconventional GVNS for Solving the Garbage Collection Problem with Time Windows", (MDPI - Open Access Publishing), Technologies 2019, 7(3), 61; https://doi.org/10.3390/technologies7030061.

[19] Papalitsas C., Andronikos T., Giannakis K., Theocharopoulou G., Fanarioti S., "A QUBO Model for the Traveling Salesman Problem with Time Windows", (MDPI - Open Access Publishing), Algorithms 2019, 12(11), 224; https://doi.org/10.3390/a12110224.

[20] Papalitsas C., Karakostas P., Andronikos T., "A Performance Study of the Impact of Different Perturbation Methods on the Efficiency of GVNS for Solving TSP", (MDPI - Open Access Publishing), Applied System Innovation 2019, 2(4), 31; https://doi.org/10.3390/asi2040031.

[21] Golub, G. H., Matrix Computations, Johns Hopkins University Press, Fourth Edition, 2013.

[22] Banerjee, S. & Roy, A., Linear Algebra and Matrix Analysis for Statistics, Chapman and Hall/CRC, 2014.

[23] Resource Description Framework (RDF), https://www.w3.org/TR/2015/NOTE-rdfa-primer-20150317/.

[24] Reynolds, D.: Position paper: Uncertainty reasoning for Linked Data. In: Workshop, vol. 14, 2014.
[25] Sistla, A.P., Hu, T., Chowdhry, V.: Similarity based retrieval from sequence databases using automata as queries. In: Proceedings of the eleventh international conference on Information and knowledge management, pp. 237–244, ACM, 2002.
[26] Schmidt M., Meier M., Lausen G.: Foundations of SPARQL Query Optimization. In: Proceedings of the 13th International Conference on Database Theory (ICDT '10), pp. 4–33, Lausanne, Switzerland, 2010.
[27] Schoenfisch, J.: Querying probabilistic ontologies with SPARQL, GI-Edition/Proceedings 232, 2245–2256, 2014.
[28] SPARQL 1.1 Query Language. Tech. rep., W3C (2013), https://www.w3.org/TR/2013/REC-sparql11-query-20130321/.
[29] Wang, X., Ling, J., Wang, J., Wang, K., Feng, Z.: Answering provenance-aware regular path queries on RDF graphs using an automata-based algorithm. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 395–396, ACM, 2014.
[30] Zhang, X., Feng, Z., Wang, X., Rao, G., Wu, W.: Context-free path queries on RDF graphs. In: International Semantic Web Conference, pp. 632–648, Springer, 2016.
[31] Gutierrez C., Hurtado C. A. and Vaisman A., "Introducing Time into RDF," in IEEE Transactions on Knowledge and Data Engineering, vol. 19, no. 2, pp. 207-218, Feb. 2007, doi: 10.1109/TKDE.2007.34.
[32] Kepner J. G., Graph Algorithms in the Language of Linear Algebra Society for Industrial and Applied Mathematics, 2011.
[33] Strang, G., Introduction to Linear Algebra, Cambridge University Press, 2016.
[34] Erisman, A. M., Reid, J. K. and Duff, I. S., Direct Methods for Sparse Matrices, Oxford University Press; 2nd edition, 2017.

