Literature Review On Computing How-Provenance For SPARQL Queries Via Query Rewriting

**Group7**

|  |  |  |
| --- | --- | --- |
| Haoyi Wang | Jianshen Wang | Xingli Li |
| A20478931 | A20491172 | A20482592 |
| Hwang203@hawk.iit.edu | jwang278@hawk.iit.edu | xli273@hawk.iit.edu |

# Introduction

# 1.1 Thesis statement

This paper introduces SPARQLprov, a novel methodology for computing how-provenance polynomials for SPARQL query solutions. The primary concepts are as follows:

1. **System Independence:** SPARQLprov utilizes query rewriting, making it independent of specific systems, ensuring compatibility across various SPARQL engines without customization.
2. **Provenance Encoding:** The approach encodes provenance annotations directly into query results and converts them into provenance polynomials using the spm-semiring model.
3. **Handling Non-Monotonic Queries:** Unlike previous methods relying on semirings, SPARQLprov computes provenance polynomials for both monotonic and non-monotonic SPARQL queries, ensuring commutation with homomorphisms.
4. **Evaluation:** SPARQLprov is evaluated using real and synthetic datasets, demonstrating reasonable processing time increases compared to original queries. It exhibits scalability and performs competitively against specialized systems like TripleProv.

**1.2 Purpose of the review and Overview:**

The review introduces SPARQLprov, addressing the absence of provenance explanations for query outcomes in existing knowledge graph query processing approaches. It discusses related work, algebraic structures for annotated data, and challenges in computing SPARQL query provenance. The paper presents SPARQLprov, emphasizing its applicability to any SPARQL engine via query rewriting. Evaluation includes algorithm phases, reification schemes, and experiments with the Watdiv benchmark dataset, highlighting reification overheads and SPARQL engine performance differences. Future work suggestions involve improving how-provenance annotations and handling aggregate queries.

**1.3 Limitation:** The paper does not explicitly address the challenges associated with implementing SPARQLprov in real-world scenarios. It emphasizes method presentation and performance evaluation on real and synthetic data.

# Sections Review

The paper introduces the origin of mappings generated by aggregation and formally defines this concept. Furthermore, it explores the process of rewriting aggregate queries and the challenges encountered when applying this method to SPARQL.

Through detailed background knowledge and related work review, the researchers introduce the concept of how-provenance, which denotes the source and generation methods of each element in the query result. They also present the concept of provenance polynomials, a formal representation method for how-provenance. Additionally, this section reviews the basic structure of SPARQL query language and the RDF data model, laying the foundation for the subsequent methods and experiments.

In the introduced method section, the researchers explain their query rewriting technique, which can transform the original SPARQL queries into queries annotated with how-provenance. They describe the steps and rules of this transformation and elucidate the process of generating provenance polynomials, enabling readers to grasp the underlying mathematical principles of their approach. The following are the detailed explanations of the techniques used in this section:

* **Query Rewriting Technique:** The authors first introduce their query rewriting technique, the core step of this method. In this process, the original SPARQL queries are transformed into a new query form capable of obtaining both query results and corresponding how-provenance information. A series of transformation rules are defined, which associate each element in the query with its source based on the structure of the SPARQL query. This association process ensures that each query result element can be accurately traced back to its source in the original data.
* **Generation of Provenance Polynomials:** The authors elaborate on how to generate provenance polynomials. Once the query is transformed into an annotated form, the authors introduce a mathematical representation, namely provenance polynomials, to depict how-provenance information. They define the structure of these polynomials as well as the meanings of the parameters within the polynomials. This polynomial form precisely and clearly describes the sources and generation methods of each query result element. Through these polynomials, the researchers achieve the goal of embedding provenance information into query results.
* **Progression of Mathematical Principles:** In this part, the authors present the progression of applying mathematical principles. They start from the original SPARQL query and gradually apply query rewriting rules to transform the query into a form annotated with how-provenance information. During this process, they employ various mathematical symbols and logical operations, such as set intersections, unions, differences, as well as polynomial addition, multiplication, and exponentiation. These mathematical principles ensure the accurate representation and transmission of how-provenance information.

The paper delves into the implementation details. The researchers provide a detailed description of the specific algorithms and optimization strategies for query rewriting. They introduce some efficient data structures and algorithms to accelerate the computation process of provenance polynomials:

* Data Structures:
  + Graph Data Structure: Since SPARQL queries typically involve multiple interconnected triple patterns, a graph data structure is used to represent the relationships between these patterns. This facilitates rapid query rewriting and computation of how-provenance information.
  + Index Structures: Various index structures (such as B-trees, hash indexes, etc.) are utilized to expedite the search for key data involved in queries, enhancing query performance.
* Algorithms:
* Query Rewriting Algorithm: The paper provides a detailed description of the query rewriting algorithm, including the order of rule application and the strategy for selecting rewriting schemes.
* Polynomial Evaluation Algorithm: For computing provenance polynomials efficiently, the authors employ advanced polynomial evaluation algorithms, avoiding naive polynomial expansion and calculation, thereby enhancing computational speed.

Moreover, this section includes an in-depth discussion of performance optimization and resource utilization, including:

* Query Rewriting Algorithm: The paper elaborately describes the specific steps and rules for transforming the original SPARQL queries into queries annotated with how-provenance. This includes identifying patterns and relationships within the queries and transforming them into a form conducive to provenance computation.
* Optimization Strategies: The researchers introduce optimization strategies employed during the query rewriting process to reduce computational complexity and enhance query execution efficiency. This may involve early-stage processing steps, such as selecting appropriate query plans or utilizing specific query optimization techniques.
* Efficient Data Structures: The paper describes efficient data structures used to store and manipulate intermediate query results. These data structures are meticulously designed to enable rapid access and manipulation of data during the provenance computation process.

The evaluation section outlines experimental evaluations of the runtime overhead and scalability of SPARQLprov. It mentions using the Watdiv benchmark RDF/SPARQL engine and measuring query execution times across different materialization schemes. It also notes setting a timeout of 300 seconds and reporting the average response time for queries executed more than five times.

The contribution section of the paper focuses on introducing a unique method for query rewriting called SPARQLprov. It provides a query solution annotated with specialized provenance polynomials designed for SPARQL queries. Additionally, it proposes a method for annotating SPARQL aggregate query solutions using lineage expressions. The conclusion of this section emphasizes extensive experimental evaluations of the runtime overhead and scalability of SPARQLprov, conducted on both real and synthetic data.

# Research method

3.1 The SPARQLprov method

The SPARQLprov method, as proposed by the authors, is a comprehensive approach designed to compute how-provenance for SPARQL queries. The method operates through three distinct stages: query rewriting, execution, and decoding, each addressing specific aspects of how-provenance computation.

1. **Query Rewriting:**

In the query rewriting stage, the original SPARQL query undergoes a transformation process. This transformation results in a new query format, capable of providing annotated answers along with how-provenance information. Crucially, this process takes into account the reification scheme utilized in the dataset, ensuring accurate association of each answer with its source and alterations. Specifically, the method employs the following algorithms and techniques to address reification-related challenges:

* 1. **Identification and Extraction of Reification:** During the query rewriting stage, SPARQLprov identifies the reification patterns involved in the original SPARQL query. Reification patterns refer to the special triples used in RDF data to represent information about other triples. By recognizing these reification patterns, the method understands which data pertains to other data, enabling accurate tracing of the origin of each answer in the how-provenance.
  2. **Application of Algorithms and Rules:** Once reification patterns are identified, SPARQLprov applies specific algorithms and rules to transform these reification patterns into a format conducive to how-provenance calculation. This may involve converting reification patterns into polynomial expressions for subsequent computations. The application of these algorithms and rules ensures the integration of reification information into how-provenance accurately.
  3. **Data Association and Matching:** After the reification patterns are transformed, the method ensures effective association between the transformed query and reified data. This might involve matching specific elements in the query (such as variables or constants) with corresponding elements in the reified data. Through this association, each answer can be linked to its reified form in the original data, providing precise source and modification details.
  4. **Polynomial Computation:** Upon obtaining answers annotated with reification information, SPARQLprov encodes this information into polynomial forms. In this process, the method might utilize an spm-semiring, a mathematical structure used for addition and multiplication operations between polynomials. This encoding ensures accurate representation of reification information and enables proper handling during computations.

By applying these algorithms and techniques, the SPARQLprov method effectively handles reified data, ensuring that each answer is accurately associated with its reified form and its source and modifications in the original data, providing precise and detailed how-provenance information.

1. **Execution:**

Following the query rewriting, the transformed query is executed on a designated storage driver, such as Virtuoso or Fuseki, using the reified dataset. The execution stage is pivotal, serving as the focal point for runtime assessment. During this phase, the method evaluates the rewritten query against the dataset, generating annotated answers that encapsulate the necessary information for how-provenance computation.The method meticulously executes several steps, as outlined in the paper, to ensure the accuracy and completeness of this information.

1. **Rewritten Query Execution:**

The rewritten query, transformed to incorporate how-provenance annotations, is executed against the designated dataset. This dataset can be hosted on storage engines such as Virtuoso or Fuseki. The execution involves interpreting the transformed query within the context of the dataset's RDF structure and triples.

1. **Data Retrieval and Annotated Answers:**

As the rewritten query is executed, the method retrieves data from the dataset based on the query's specifications. The retrieved data includes not only the query results but also additional information derived from the reification patterns within the dataset. This additional information is crucial as it helps in understanding the sources and alterations associated with each answer.

1. **Incorporation of Reification Data:**

During the execution, the method takes into account the reification scheme employed in the dataset. Reification patterns, representing metadata about triples in RDF data, are utilized. These patterns are decoded and incorporated into the annotated answers. By incorporating reification data, the method ensures that each answer is not merely a result but a comprehensive entity, including its origin and any transformations it underwent.

1. **Generation of How-Provenance Polynomials:**

The annotated answers obtained during the execution phase are then further processed to generate how-provenance polynomials. These polynomials serve as formal representations of the how-provenance information associated with each answer. The generation of these polynomials involves mathematical computations based on the reification data and the original query structure.

1. **Accuracy and Completeness:**

Throughout this phase, the method focuses on ensuring the accuracy and completeness of the annotated answers. By correctly interpreting the reification patterns, associating them with query elements, and encoding them into polynomials, the method guarantees that the how-provenance information encapsulated in the annotated answers is precise, detailed, and reflective of the data's actual sources and alterations.

In summary, this phase of the SPARQLprov method is a meticulous process where the rewritten query is executed against the dataset, annotated answers are generated by incorporating reification data, and how-provenance polynomials are formulated. The method's effectiveness lies in its ability to seamlessly integrate reification information into the annotated answers, providing a robust foundation for precise and comprehensive how-provenance computation.

1. **Decoding:**

In the decoding stage, a pivotal component of the SPARQLprov method, the annotated answers acquired from the execution phase undergo a meticulous transformation process. This transformation involves converting these annotated answers into how-provenance polynomials, which serve as intricate mathematical representations providing nuanced insights into the origins and modifications that led to the formation of each answer. This process holds paramount importance as it translates raw data into comprehensive and understandable how-provenance information, enhancing the transparency and interpretability of the results.

* 1. **Annotated Answers Processing:**

Initially, the annotated answers obtained during the execution phase are meticulously processed. These answers contain not only the query results but also additional information derived from the reification patterns within the dataset. This supplementary information includes details about the sources, transformations, and alterations associated with each answer.

* 1. **Conversion into How-Provenance Polynomials:**

The processed annotated answers are then subjected to a sophisticated mathematical transformation. This transformation involves converting the intricate details encapsulated in the annotated answers into structured how-provenance polynomials. These polynomials are formulated using precise mathematical operations, which are based on the reification data, the original query structure, and the specific transformations applied during query execution.

* 1. **Detailed Insights into Origins and Modifications:**

The resulting how-provenance polynomials are not mere mathematical representations but encapsulate detailed insights into the origins and modifications contributing to each answer. These polynomials are designed to capture the intricacies of the data's evolution, providing a granular view of the sources from which the answer originated and the series of alterations it underwent.

* 1. **Comprehensive and Understandable Information:**

By converting the annotated answers into how-provenance polynomials, the decoding stage ensures that the how-provenance information is not only comprehensive but also highly understandable. Each polynomial acts as a comprehensive record, detailing the entire journey of an answer, making it clear and interpretable for researchers, analysts, and other stakeholders.

* 1. **Enhancing Transparency and Interpretability:**

The transformation into how-provenance polynomials enhances the transparency and interpretability of the how-provenance information. Instead of presenting raw, complex data, the method provides structured, polynomial representations. This structuring simplifies the information, making it accessible to a wider audience and facilitating a deeper understanding of the data's lineage.

In essence, the decoding stage plays a fundamental role in the SPARQLprov method by converting raw annotated answers into sophisticated how-provenance polynomials. These polynomials offer detailed, understandable, and interpretable insights into the origins and modifications associated with each answer, significantly enhancing the transparency and clarity of the how-provenance information.

**3.2 Key Methodological Enhancements:**

* **Extending K-annotated SPARQL Algebra:**

The method extends the K-annotated SPARQL algebra, adapting it to handle missing operators. It caters to both monotonic and non-monotonic operators but excludes queries involving aggregation.

* **Addressing Challenges:**

The paper tackles the challenges associated with encoding polynomials in an spm-semiring for both monotonic and non-monotonic operators. For aggregate queries, a specialized rewriting approach is employed, computing aggregate function values and polynomial explanations separately. These results are then combined using a strict compatibility notion. In the case of non-monotonic operators, an encoding and query rewriting technique based on spm-semirings is proposed, facilitating the computation of how-provenance.

By innovatively integrating these stages and addressing specific challenges, the SPARQLprov method provides a robust framework for computing how-provenance in SPARQL queries, enhancing the understanding of query results in the context of their origins and alterations.

# Evaluation

## Discuss experimental settings and datasets for evaluation

The experimental settings and datasets for the evaluation of SPARQLprov are described in the provided information. Here is a summary:

1. **Storage Driver:** The evaluation used Virtuoso as the storage driver to test SPARQLprov.
2. **Timeout and Execution:** A timeout of 300 seconds was set for all experiments. The average response time of the queries was measured over 5 executions after a warm-up phase.
3. **Dataset:** The Watdiv benchmark was used to generate synthetic datasets of different sizes. The benchmark offers 20 query templates divided into four categories: linear queries (L), star queries (S), snowflake-shaped queries (F), and complex queries (C). Additionally, 5 new query templates were introduced to account for non-monotonic queries (O).
4. **Reification Schemes:** The evaluation measured query execution times on three popular reification schemes: named graphs, Wikidata, and standard reification. A copy of the data without reification was also used to determine the overhead caused by the reification schemes.
5. **Query Execution Breakdown:** The execution time of a provenance query was broken down into three components: the baseline share of executing the original query on non-reified data, the reification overhead of executing the query over the reified data, and the provenance overhead of computing the provenance query over the reified data.
6. **Real Data:** SPARQLprov was also evaluated on the RDF Wikidata dump, which contained 942M relationships encoded with the Wikidata reification scheme. Three types of queries (star, union, and minus) were generated and the corresponding provenance queries were computed.
7. **Comparison with Other Systems**: SPARQLprov was compared with TripleProv, a specialized how-provenance solution for SPARQL, and GProM, a system to compute how- provenance for SQL queries. The comparison included runtime evaluation and qualitative evaluation of the computed annotations.

### 4.2 Summarize key results related to performance overhead and comparison with other systems

The provided discourse examines the fundamental outcomes associated with performance overhead and compartions them with other frameworks. Presented below is a succinct summary of the principal discoveries:

**1.Performance Overhead:** The analysis of SPARQLprov on real-world datasets demonstrates that it engenders an average provenance overhead of 36% for star queries, 43% for union queries, and 56% for minus queries in Virtuoso. In Fuseki, the overhead stands at 12% for star queries, 19% for union queries, and 30% for minus queries. For non-monotonic minus queries, the overhead is greater due to the computation of explanations for a substantial number of bindings(Figure 4-1 ).

**2.Comparison with TripleProv:** SPARQLprov surpasses TripleProv, a specialized how- provenance solution for SPARQL, in terms of runtime for most queries. However, TripleProv performs superiorly for specific star queries without constants by virtue of its architecture predicated on star patterns known as molecules(Figure 4-1).

**3.Comparison with GProM:** SPARQLprov also outperforms GProM, a system for computing how-provenance for SQL queries, in terms of runtime. Despite the utilization of an efficient RDF representation by GProM, the discrepancy in runtime is generally less than 0.5 seconds(Figure 4-2).

**4.Impact of Reification Schemes**: The evaluation results indicated that different reification schemes had an impact on performance. The standard reification scheme introduced more triples, leading to increased complexity and performance overhead during query execution. Other reification schemes showed relatively lower performance overhead.

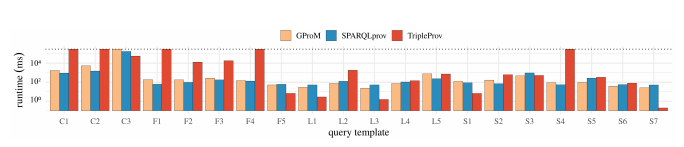
**5.Influence of Query Type:** The experiments do not disclose any noteworthy influence of the query type on performance. Instead, performance is influenced by the sizes of results, the selectivity of triple patterns, and the presence of non-monotonic operators, which entail greater evaluation costs.

**6.Scalability and Aggregation:** The evaluation of SPARQLprov on the TPC-H benchmark reveals that runtime grows proportionally with data size. Virtuoso consistently outperforms Fuseki, albeit the overhead of aggregation in base queries is more conspicuous for

Virtuoso. The runtime overhead of provenance is unaffected by the scale factor and the number of answers in Fuseki, while in Virtuoso, it is primarily contingent upon the query template.



**Figure 4-1: Watdiv data overhead. (**Cited from the paper Figure4**)**



**Figure 4-2 Watdiv query execution times per engine and query template. The top dotted line represents the timeout.**

**(**Cited from the paper Figure 5**)**

# Discussion

* 1. **The advantages and disadvantages of SPARQLprov method**

1. **The SPARQLprov method presents various advantages.**

Firstly, it is designed to be system-agnostic, allowing for easy integration into existing SPARQL engines without requiring customized extensions. This enhances the capabilities of SPARQL engines across different systems without the need for significant modifications.

Secondly, the research evaluates the runtime overhead of SPARQLprov on both real andsynthetic data. The results demonstrate that SPARQLprov incurs a reasonable runtime overhead, with an average provenance overhead ranging from 12% to 56% for different query types. This indicates that SPARQLprov is capable of computing how-provenance with acceptable performance impact.

Thirdly, the evaluation of SPARQLprov on a large real-world dataset, namely the RDF Wikidata dump, showcases its scalability. The approach is able to handle datasets with 942 million relationships encoded with the Wikidata reification scheme. This scalability is crucial for efficiently managing large-scale knowledge graphs and complex queries.

Lastly, in terms of runtime evaluation, SPARQLprov is compared with other systems such as TripleProv and GProM. The results reveal that SPARQLprov outperforms TripleProv in most cases and competes well with GProM. This highlights the efficient runtime performance of SPARQLprov while remaining system-agnostic.

**b.However, there are also disadvantages to the SPARQLprov method.**

Firstly,The SPARQLprov method uses queries with non-monotonic operators that are more expensive and require more computational resources. This may mean that the SPARQLprov method may perform worse when dealing with large data sets or complex queries when dealing with non-monotone operators.

Secondly,The description mentions that the SPARQLprov method does not include queries with aggregate functions. This limits the scope of the approach because many real-world queries involve aggregation operations.

Lastly, the evaluation of SPARQLprov primarily focuses on monotonic queries and excludes queries with optional and diff operators. This limitation arises from the underlying semirings framework used by TripleProv and GProM, thus leaving the performance and effectiveness of SPARQLprov for non-monotonic queries unexplored in this research.

In conclusion, while SPARQLprov offers several advantages such as system-agnosticism, efficient runtime overhead, scalability, and competitive performance, it also has limitations related to aggregate queries, limited comparison with other systems, and the exclusion of non- monotonic queries in the evaluation.

## Possible expansion or open issues for future work

1. The current iteration of SPARQLprov lacks support for aggregate queries, which presents an opportunity for valuable expansion. Enhancing the methodology to accommodate aggregate functions such as COUNT, SUM, AVG, among others, would greatly contribute to a more comprehensive analysis of SPARQL queries.
2. The focus of this research lies in examining monotonic queries, while excluding non- monotonic queries involving optional and diff operators. However, exploring thecomputation of how-provenance for non-monotonic queries and integrating them into the SPARQLprov methodology would open up an interesting avenue for future investigation.
3. Although the research provides a comparative analysis of SPARQLprov in relation to TripleProv and GProM, further comparisons with other provenance systems such as Perm and HUKA would yield a more comprehensive understanding of the strengths and weaknesses of SPARQLprov.
4. It would be beneficial to explore techniques aimed at optimizing the runtime performance of SPARQLprov, particularly for complex queries and large-scale knowledge graphs. Potential avenues for optimization could involve query rewriting, caching, or parallel processing.
5. A valuable direction for future work would involve investigating the integration of SPARQLprov into existing SPARQL engines, such as Apache Jena or Virtuoso. Such integration would enhance the practical usability and adoption of SPARQLprov within the wider SPARQL community.
6. Gaining insights into the effectiveness and performance of SPARQLprov in practical scenarios can be achieved by conducting evaluations on real-world use cases and datasets. This evaluation process could include analyzing the impact of SPARQLprov on different types of queries and datasets.
7. In order to facilitate wider adoption and exploration of SPARQLprov by researchers and practitioners, there is a need to develop user-friendly interfaces or tools that simplify the adoption and usage of SPARQLprov.

# Conclusion

This paper endeavors to offer explanations regarding the provenance of query results within the context of extensive knowledge graphs that are constructed from multiple sources. The research introduces a novel contribution in the form of the proposed technique named SPARQLprov, which relies on query rewriting and is independent of any specific system. This implies that it can be applied to standard SPARQL engines without the need for customized extensions. SPARQLprov introduces the concept of how-provenance, which expands upon the existing notions of lineage and why-provenance. The representation of how-provenance is achieved through the use of polynomials in a commutative semiring, allowing for structured explanations of the sources and processes that contribute to query results.

The significance of this research lies in its provision of a method for computing how-provenance that can be applied to a broad array of SPARQL engines and can accommodate both monotonic and non-monotonic queries. Additionally, it permits lineage annotations for queries with aggregates. The research includes an evaluation of the approach using both real and synthetic data, which demonstrates a reasonable runtime overhead when compared to the original query. Furthermore, it demonstrates competitiveness with state-of-the-art solutions for how-provenance computation.