SPARQL Optimization Using Re-ordering Joining Patterns with Surrogate Key Concept and Subset Patterns

Rupal Gupta[1,2,*] and Sanjay Kumar Malik[1]

[1] *USIC&T, Guru Gobind Singh Indraprastha University, Dwarka, Delhi, 110078, India*
[2] *College of Computing Sciences and IT, Teerthanker Mahaveer University, Moradabad, Uttar Pradesh, 244001, India*
E-mail: rupal.gupta07@gmail.com; skmalik@ipu.ac.in
[*] *Corresponding Author*

Received 21 September 2023; Accepted 31 March 2024

Abstract

Semantic web data resides on the web in the form of knowledge graphs known as RDF graphs and searching around the web has been always a crucial task. For the data retrieval of RDF data of the semantic web, SPARQL query language has been used which in turn is based on triple patterns and joins. Optimization of SPARQL query has been a problematic concern for decades due to the large amount of triple patterns associated with RDF data. Although several researchers have put a lot of effort into the optimization of SPARQL query, it is difficult to understand the concept from scratch due to its diversified nature. This paper analyses various optimization techniques for the SPARQL query used with the semantic web to process knowledge graphs. These techniques include join-based, heuristic-based, rule-based, and indexing-based approaches for optimization. This paper will help researchers in this domain to easily get into the core concept of SPARQL execution along with various optimization approaches used for query processing, which can help in various other domains like linked open data and information retrieval. In this paper, an optimization algorithm HSOA (hybrid SPARQL optimization algorithm) has been proposed, which comprises the features of index-based, cost-based, and triple reordering-based optimization approaches. The proposed hybrid algorithm has been designed specifically for n-triple RDF data, which comprises subset patterns, and surrogate key concepts. The results produced by the proposed algorithm are encouraging and have also been tested and compared with the benchmark dataset and SPARQL queries like LUBM, BSBM, and SP2Bench.

Keywords: SPARQL, RDF, optimization, indexing, reordering, metaheuristics, triple patterns.

1 Introduction

Optimization in the semantic web is of great concern due to its data representation formats and flexibility features. In the semantic web, SPARQL query processing has a great role in integration as well as for information processing from RDF data and ontology. The rapid growth of RDF data makes it too complex to analyze and process after the summarization of the data [1]. Processing needs optimization to a certain extent for better information retrieval from RDF data. Optimizing information retrieval is a key step, especially when the dataset is large. Optimization in this domain can be categorized with different techniques based on the indexing concept used with RDF data, along with processing-based focusing SPARQL query optimization. This paper discusses the different aspects of optimization that can take place on indexing or triple ordering and joining concepts among triples used with RDF data and SPARQL query. This paper also proposes a triple ordering-based optimization algorithm that has been applied to different RDF datasets. The result obtained by the proposed algorithm gives better results in information retrieval as compared to the earlier approaches. The results have also been compared with other benchmark datasets and SPARQL queries like LUBM, BSBM, and SP2Bench.

The major contributions are as follows:

- An algorithm HSOA (hybrid SPARQL optimization algorithm) is proposed while merging various concepts of optimization based on index-based and query-based (reordering), and may also be utilized with processing-based techniques.

- The dataset used in the paper can be treated as a benchmark of the RDF dataset used for applying machine learning algorithms.
- The queries used in the experiment results can be used to integrate semantic web and machine learning algorithms to produce further results that can be explored with knowledge graphs.
- The proposed algorithm works well with the existing benchmark datasets and queries that focus on the performance criteria based on triple pattern ordering.

This paper has been divided into four sections. Section 1 gives an introduction to semantic web technologies along with the motivation for SPARQL query optimization. Section 2 investigates various techniques used for SPARQL optimization. Section 3 explains the workflow of the proposed hybrid SPARQL optimization technique and the last section interprets the experimental setup, results, and comparison with benchmark datasets and queries.

2 Semantic Web

The semantic web, also known as Web 3.0, is an extension of Web 2.0 that focuses on a web with intelligence. The intelligence within this web is that the machines and devices can interpret and understand the meanings behind sentences during information processing over the web [2]. The data over the web can be treated as an interconnection for an enormous amount of data having attributes and relationships among them. The interconnectedness of data makes semantic queries more useful for getting the required knowledge from the huge amount of data that resides in the form of RDF over the web (semantic web). For example, before the internet, in early documents citations were followed to get the data that may or may not be available to everyone [3].

Today, searching for specific knowledge will also bring information that in return may not be related to the search, but with the help of semantic web technologies useful information will also be retrieved. It makes computers understand and process the meaning by the interpretation of data in a similar way humans interpret things to achieve their goals [4]. These kinds of capabilities exist in the current era web for smart meaningful searching. Knowledge graphs are playing a vital role which are in terms of RDF graphs used in the semantic web for data representation. SPARQL queries are used to process and search the data from these huge interconnected data sources over the web known as the web of data.

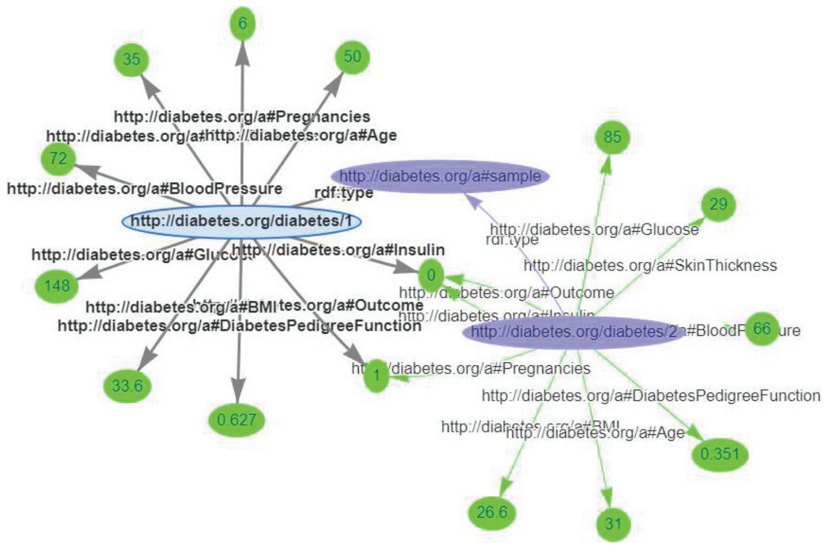

6
35
50
http://diabetes.org/a#Pregnancies
http://diabetes.org/a#http://diabetes.org/a#Age
72
http://diabetes.org/a#BloodPressure
rdf:type
http://diabetes.org/diabetes/1
http://diabetes.org/a#Insulin
148
http://diabetes.org/a#BMI
http://diabetes.org/a#DiabetesPedigreeFunction
33.6
0.627
http://diabetes.org/a#sample
85
29
http://diabetes.org/a#Glucose
http://diabetes.org/a#SkinThickness
http://diabetes.org/a#Outcome
0
1
http://diabetes.org/diabetes/2
66
http://diabetes.org/a#DiabetesPedigreeFunction
http://diabetes.org/a#Age
0.351
26.6
31

Figure 1 Sample RDF graph of two instances of the Diabetes RDF dataset.

2.1 RDF (Resource Description Framework)

RDF, which stands for resource description framework, is a standard developed by the World Wide Web Consortium (W3C). It is a specification created originally for metadata data modelling [5] and is now treated as a general method used for information modelling or a conceptual description that is enforced in web resources, using formats of data serialization and a variety of syntax notations.

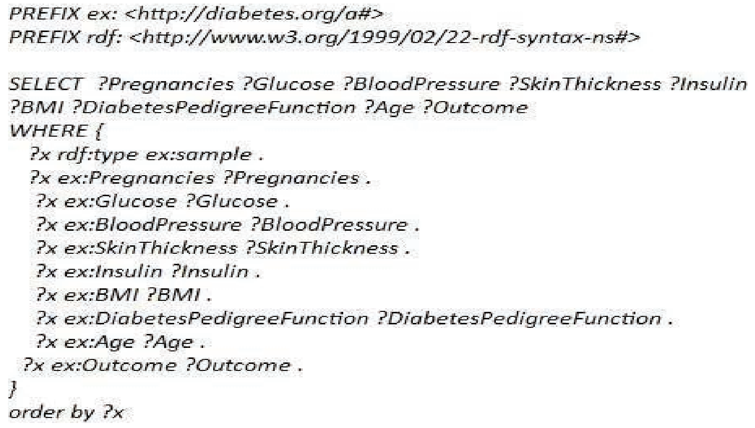
The data model of RDF is similar to a classical conceptual modelling approach (such as ER-diagram or class diagram) which makes it flexible and scalable. The idea is based on making statements about resources (being particular web resources) expressions called triples [6]. The name triples are given because they follow a structure of subject-predicate-object relationship structure. The subject defines the resource, while the predicate defines traits or features of the resources, and conveys the relationship between the subject and the object [7]. With various serialization formats (i.e., file formats). RDF is an abstract model, so the specific encoding varies from format to format for triples or resources. The RDF data can be easily viewed as RDF graphs, one sample Diabetes RDF dataset with two instances has been shown in Figure 1, which has been used for further SPARQL query processing and optimization in this paper.

2.2 SPARQL (Simple Protocol and RDF Query Language)

Simple protocol and RDF query language (SPARQL) is a general-purpose query used to process data, based on basic graph patterns (BGPs) [8]. It works on the triple ordering of patterns placed in a query. In this paper, different queries have been used to find optimal queries on different RDF data. This query can be better understood further using SPARQL algebra (internal evaluation structure for processing) and graph visualizations of checking its types like star, chain, and others. The SPARQL engine in a query treats each triple pattern as a separate scan, therefore the sequence of the scans is crucial and is needed to be smarter to choose the next scan in the order. A sample SPARQL query which has been optimized by the proposed optimization algorithm is shown in Figure 2. The SPARQL query retrieves the information for RDF data using triple patterns used after the where clause of the query.

2.3 OWL (Ontology Web Language)

Web ontology is another useful rich taxonomy of data representation in the semantic web. This is also one of the standards of the semantic web and also a major technology used to process the data [9]. The focus area of semantic web ontology is the knowledge representation of the things that can be easily published and understood by the machine over the web. Ontology can be used to easily describe relationships and concepts among entities. One sample LUBM benchmark ontology for university [10] has been represented using an intermediate snapshot in Figure 3.


```
PREFIX ex: <http://diabetes.org/a#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT  ?Pregnancies ?Glucose ?BloodPressure ?SkinThickness ?Insulin
?BMI ?DiabetesPedigreeFunction ?Age ?Outcome
WHERE {
   ?x rdf:type ex:sample .
   ?x ex:Pregnancies ?Pregnancies .
    ?x ex:Glucose ?Glucose .
    ?x ex:BloodPressure ?BloodPressure .
    ?x ex:SkinThickness ?SkinThickness .
    ?x ex:Insulin ?Insulin .
    ?x ex:BMI ?BMI .
    ?x ex:DiabetesPedigreeFunction ?DiabetesPedigreeFunction .
    ?x ex:Age ?Age .
  ?x ex:Outcome ?Outcome .
}
order by ?x
```

Figure 2 Sample SPARQL query used on the diabetes RDF dataset.

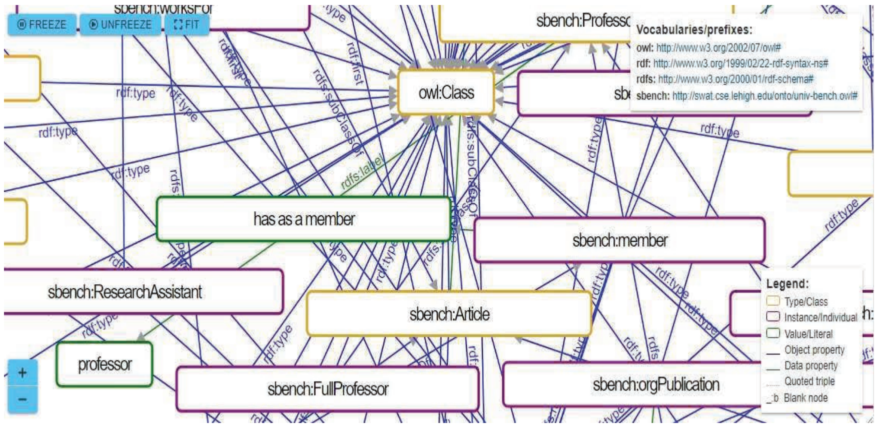


Figure 3 A university ontology LUBM benchmark dataset showing classes and properties [10].

3 Literature Review

Groppe [11] proposes an index construction approach to generate a mapping for RDF from unique identities and evaluates indices according to the collation order of RDF based on subject, predicate, and objects. Further, it can be useful for data analytics. The paper defines steps to indices and represents a comprehensive experiment of the proposed model for execution time analysis by importing 1 billion triples [11]. Nguyen et al. [12] represent the challenges of storing and retrieving RDF data efficiently with graph-based and relational-based approaches. They also focus on RDF data indexing, where a structure index was used to obtain RDF data for evaluating query patterns based on the execution plan, reducing intermediate triples results that are not used for joining further to obtain the result. It has been seen that the system reduces the query execution time by up to 79% compared with the conventional query processing approach like that used by Jena [12].

Wu et al. [13] propose effective heuristic methods with generic data partitioning approaches that can consider a much larger space with the same search time. It has been presented that heuristics rules can be used to reduce the degree of join variables along with the number of triples in the join graph to get better performance [13]. Chawla et al. [14] propose that the SPARQL query execution problem can be visualized as a graph traversal. A modified all-pair shortest path algorithm was analyzed with a complete SPARQL query graph and they found an optimal order of execution using heuristics for SPARQL which is obtained with minimum cost. It has been observed that the Floyd Warshall algorithm works faster than the Johnson algorithm in computing the execution plan [14].

Papailiou et al. [15] present the H2RDF+ Tool, which is a fully distributed RDF store capable of storing and querying large RDF datasets. A scalable adaptive decision about centralized and distributed join execution is presented and effective results are being found in the Hadoop environment using HBase indexes [15]. Oh et al. [16] present an efficient query processing system that uses a job-optimized, map-only query planner. They conclude that by utilizing a careful design HBase storage schema, RDF data can be input to the map phase so that reordering is not needed to evaluate the query. They found that by utilizing abstract RDF data to find out which pattern the result lay in, the number of inputs to map-side jobs could be reduced, resulting in better performance [16].

Ge et al. [17] propose a query decomposition algorithm that has further been used in a source selection optimization strategy, which has been proved efficient in reducing the number of remote requests. Here a query execution plan has been proposed for efficient evaluation of top-k queries. The proposed algorithm can be applied to top-k queries for single variable ordering as well as expression ordering [17]. Li et al. [18] perform a mapping of keywords to draw RDF graphs over federated queries in the proposed model. The utilization of a full-text search has been done to map keywords to schema graphs which has been further used to generate SPARQL queries. Further, to optimize the query evaluation process, the technique of query rewriting has been proposed [18]. Peng et al. [19] present a study on a federated RDF system with optimization multi-query evaluation. The proposed algorithm finds common sub-queries with an effective cost model. The query generated has been further rewritten into smaller queries. Also, a discussion on the efficient selection of sources and join order has been presented. For the experiments standard and new operations of SPARQL 1.1 have been used [19].

Jose and Poulose [20] discuss the difficulties encountered when conducting query joins, which involve merging data from sources to improve the efficiency of web systems. They introduce an optimization strategy that utilizes a customized version of the grey wolf optimization (GWO) algorithm to enhance query join operations. The GWO process involves three stages: encircling, hunting and attacking. Their study demonstrated that implementing GWO on the LUBM 10000 dataset reduced execution time by 2–5 minutes and improved precision scores such as recall and f measure, by 2–7%. The GWO algorithm, inspired by grey wolf hunting behaviour aims to minimize the cost of query execution through join procedures in a query model join optimization on the semantic web. Additionally, they proposed a modified GWO method specifically tailored for query join optimization to navigate semantic web data complexities. The results indicate that their adapted GWO algorithm outperforms other methods, by optimizing joins and enhancing query execution speed [20].

Dhiman et al. [21] introduced the emperor penguin optimizer (EPO), a novel swarm-based metaheuristic algorithm inspired by the huddling behavior of emperor penguins. The EPO algorithm is applied to solve six real-life constrained and unconstrained engineering design problems [21]. They also introduced the seagull optimization algorithm (SOA) [22], a bio-inspired metaheuristic algorithm designed for solving computationally expensive problems. The algorithm is inspired by the migration and attacking behaviors of seagulls in nature, which are mathematically modeled to emphasize exploration and exploitation in search spaces [22]. They introduced in another research paper the spotted hyena optimizer (SHO) [23], a novel swarm-based metaheuristic algorithm inspired by the social hierarchy and hunting behavior of spotted hyenas. The algorithm consists of three key steps: searching for prey, encircling and attacking, which are mathematically modeled and implemented. The SHO algorithm is compared with eight existing metaheuristic algorithms on 29 benchmark test functions, showcasing superior performance in terms of convergence and computational complexity [23].

Kaur et al. [24] introduced the tunicate swarm algorithm (TSA), a bio-inspired metaheuristic optimization approach mimicking the jet propulsion and swarm behaviors of tunicates during navigation and foraging. The algorithm is compared with other metaheuristic approaches and applied to engineering design problems, showcasing its robustness and ability to generate superior solutions. The TSA is inspired by the unique jet propulsion and swarm behaviors of tunicates, marine organisms that emit bioluminescent light and migrate vertically in the ocean. The algorithm is described mathematically, and its efficacy is tested on various benchmark functions, outperforming other algorithms in terms of optimal solution generation [24]. Dhiman et al. [25] introduced the binary emperor penguin optimizer (BEPO), an extension of the emperor penguin optimizer (EPO), a metaheuristic algorithm inspired by the huddling behavior of emperor penguins. The study evaluates BEPO's effectiveness using 25 benchmark functions and compares it with seven other binary metaheuristic algorithms. The paper also applies BEPO to feature selection problems on 12 benchmark datasets, demonstrating its superior performance and robustness compared to other algorithms [25].

Dehghani et al. [26] introduced the darts game optimizer (DGO), designed to simulate the rules of a darts game. The key concept is to maximize points scored by players in their throws at the game board. DGO's performance is compared to existing algorithms, showcasing its exploration and exploitation capacities [26]. Dehghani et al. [27] also introduced a multileader optimizer (MLO) designed to address optimization problems by employing multiple leaders to guide the population toward optimal solutions. MLO's performance is evaluated on 23 standard objective functions and compared with eight existing optimization algorithms [27]. Dehghani et al. [28] introduces the binary orientation search algorithm (BOSA), a binary model derived from the orientation search algorithm (OSA). The performance of BOSA is compared with eight other algorithms on twenty-three benchmark test functions, demonstrating its high ability to solve optimization problems [28].

For optimization problems, various meta-heuristics approaches like STOA [29], ESA [30] and RSO [31] may also be utilized. SPARQL optimization can also be correlated with the existing methodology of meta-heuristics and bio-inspired techniques.

4 Motivations

In the semantic web, SPARQL query optimization has been a major concern for decades due to its complex nature of processing, which is based on triple-ordering joins [32]. The complexity of processing especially the large RDF requires lots of join-patterns among the triple form of RDF based on (S, P, O) subject, predicate, and object.

The major contribution of this paper towards the semantic web is that the proposed optimized algorithm provides a hybrid approach of SPARQL optimization which is based on subset patterns and uses the logic of surrogate keys for the introduction of new triples with unique object values, i.e., generate an index on a specific triple. The hybrid approach has not been seen yet. The RDF dataset used in the paper is based on the UCI repository which is a standard repository of data for researchers [33, 34]. The RDF data used in the experimental setup can also be used as a standard, focusing machine learning applications on it. The methodology used in the paper will help in the integration of machine learning on semantic web data, where SPARQL query can be used to process it efficiently.

As per the literature review, optimization of the SPARQL query can be categorized into three specific approaches mentioned below:

4.1 Index-based Approach

Index-based optimization is based on the indexes used on RDF data. RDF data is based on triple patterns of subject (s), predicate (p), and object (o). The information retrieval is based on the triple pattern joins used with SPARQL queries to fetch the information retrieval from stored RDF data in any data store or database on any platform like the Jena Fuseki server [35], Stardog [36], etc. Any triple pattern can be joined depending on the format and structure of RDF data. For a better understanding of the data and its processing through SPARQL, analysis of RDF data and SPARQL query joining patterns is required. As per the complexity of the SPARQL joining pattern, it is always better to have an index on either field used in joining patterns. This indexing can be done by assigning a unique identifier to every subject, predicate, or object [37].

4.2 Query-based Approach

This optimization is based on a query-based approach, which focuses more on the patterns used in the SPARQL query. The SPARQL query can work in various shapes like star, chain, and others. These query shapes emphasize the joining patterns of triples used in filtering the data. The query-based approach broadly works with either rewriting the rule-based or focusing on reordering of joining patterns used in SPARQL queries. This approach requires knowledge of SPARQL execution plans and the shapes used to filter the data. The SPARQL graphs can be explored using joining patterns of triples used with queries [38]. The execution plan of the SPARQL queries can also be analyzed to find the sequence of the operations used for information retrieval.

4.3 Processing-based Approach

Processing-based optimization focuses on the processing-based aspects in terms of parallel processing capabilities to filter the data. This approach is majorly used with distributed database systems where RDF data is stored. These data are very large RDF graphs that are spread over different URIs over the web. This approach also focuses on fast processing capabilities which can be utilized with map-reduce/Apache spark/Hive [39, 40].

In this section, the categorization is made and the analysis of different approaches is represented. The categories in which the SPARQL optimization can be put are shown in Table 1. According to the literature survey, SPARQL optimization can be categorized as query-based, storage-based, or processing-based. Query-based optimization takes place as per the written queries and its sequence of operations like filtering, selection, and joins. Storage-based optimization is based on the indexes made on the triple pattern of subject, predicate, and object. Processing-based optimization is categorized based on processing frameworks based on processing speed and storage. Concepts like parallel processing and advanced processing concepts with map-reduce and Apache Spark have been used in this category.

5 Proposed Methodology for SPARQL Optimization

The optimization algorithm proposed in this paper is a hybrid approach primarily based on index-based and query-based approaches. The algorithm can also be utilized with the better processing capabilities of processing-based optimization techniques. The nobility of the optimized algorithm is that it is a hybrid optimization technique that has the capabilities of three approaches of optimization, that is query query-based, index-based, and processing-based. This hybrid approach produced better results and output for generating optimized queries for better information processing. This optimized SPARQL query can be further utilized in various ways for information processing, and integration of the semantic web with other latest technologies of computer science that use SPARQL for integration. SPARQL queries have had a variety of roles in semantic web engineering. Due to its complex nature of processing it is always better to produce effective optimized SPARQL queries so that efficient results can be produced.

There can be various ways to optimize the overall result obtained while applying the SPARQL query on RDF also known as knowledge graphs in some sense. The optimization can be of either type like eliminating unnecessary triple ordering joins during the evaluation, storing and using the pre-aggregated computation when and where it can be used to reduce the cost, optimize the data despite optimizing the Query. In this section, an algorithm has been proposed which is based on the hybridization of a few optimizations mentioned in Section 4.

Table 1 Description and comparison of SPARQL optimization techniques

Optimization Category	Key points [description]	Query Based (q)/ Storage Based (s)/ Processing Based (p)
Index based	In this technique, indexes have been constructed using some indexing schemes like the RP index or RG index. These indexes are based on a triple pattern that is further used with SPARQL queries with filtering and joins. Indexing-based approaches produce faster results this is how the query optimization takes shape.	(s)
Re-writing rules	This is rewriting rules while analyzing the operator tree generated from the SPARQL query. The analysis can also be done using the generation of the query execution plan. In general, the set of rules has been applied to optimize the operator tree until the rules do not apply to the tree. The relational algebra for SPARQL has been visualized to put projection and selection operations before the execution of different types of joins, wherever it can be applied.	(q)
Join based	Different types of joins are used for SPARQL query optimization. The order of joining the triple patterns in different shapes of query matters a lot for making an efficient execution plan. Triple join re-ordering has been made to optimize the solution [41]. Little join-based exploration is needed like NaryJoin, NestedLoopJoin, MergeJoin, etc.	(q) + (p)
Heuristic rules	This is another reordering-based optimization technique that uses meta-heuristic approaches like ACO, PSO, etc. These approaches solve the optimization problems using one objective function and use further iterations to get the optimized result [7].	(q) + (p)
Parallel processing	This optimization approach is based on parallel processing frameworks like Spark and MapReduce. Processing-based tools like S2RDF and H2RDF are widely used to process large RDF graphs efficiently with the advanced processing capacities of Hive, HBase, and Apache Spark [15, 32, 42].	(p)

5.1 Proposed Workflow for SPARQL Optimization on RDF

In this section, the proposed HSOA (hybrid SPARQL optimization algorithm) has been represented using the execution workflow using Figure 4. The execution workflow can be viewed from the top left corner to the bottom right corner mentioned as blocks with sequential numbers. The HSOA will work with all those RDF datasets where there is non-existence of any candidate triple pattern in it. If the RDF dataset already contains it, then directly algorithm_Optimizer can be used to produce the optimized SPARQL query. This model has been specially constructed where SPARQL queries are used repeatedly on RDF datasets. One such example is applying machine learning models to train the model and further test the model with different variants.

The execution workflow has been represented using a block diagram in Figure 4 which contains sequential numbers. A detailed description has been elaborated using Table 2.

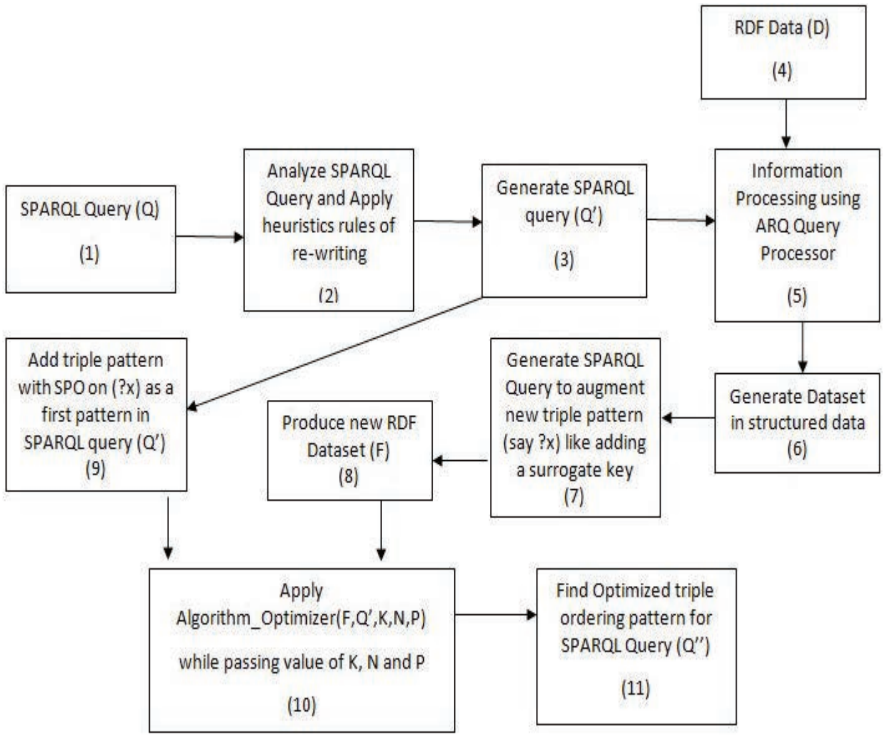

RDF Data (D) (4)
SPARQL Query (Q) (1)
Analyze SPARQL Query and Apply heuristics rules of re-writing (2)
Generate SPARQL query (Q') (3)
Information Processing using ARQ Query Processor (5)
Add triple pattern with SPO on (?x) as a first pattern in SPARQL query (Q') (9)
Generate SPARQL Query to augment new triple pattern (say ?x) like adding a surrogate key (7)
Generate Dataset in structured data (6)
Produce new RDF Dataset (F) (8)
Apply Algorithm_Optimizer(F,Q',K,N,P) while passing value of K, N and P (10)
Find Optimized triple ordering pattern for SPARQL Query (Q'') (11)

Figure 4 Execution workflow of HSOA (hybrid SPARQL optimization algorithm).

Table 2 Functioning description of workflow of HSOA (hybrid SPARQL optimization algorithm)

Sequence Step	Description	Working/ Functioning	Next Sequence Step to Follow (Output Which Will be Input to the Further Process)
(1)	SPARQL query (Q)	This is an input query given to process the data	(2)
(2)	Analyse SPARQL and apply re-writing rules on it.	Apply the heuristics and re-writing rules on the SPARQL query. This process involves the selection and projection to be applied earlier before applying the triple-ordering join patterns.	(3)
(3)	Generate SPARQL query (Q')	This is a newly updated query that has been applied to the rules of step 2.	(5)
(4)	RDF Data (D)	This is an input RDF data that contains the URI source and triple patterns in the form of a subject, predicate, and an object in n-triple form.	(5),(9)
(5)	Information processing using ARQ query processor	In this step, the SPARQL query has been applied to retrieve data from the RDF dataset. Further, any tool can be used to process this step. This step involves the JENA ARQ (an RDF query) engine for query processing [31]. Further, any tool like JENA ARQ, Twinkle, SPARQL DBpedia endpoint, JENA Fuseki Server, Stardog, etc., can be used to apply the step [35, 36, 44].	(6)
(6)	Generate dataset in structured data	Mostly semantic web data is in semi-structured form. It is helpful to produce the dataset in a structured form. In the implementations of this step, the dataset is generated in CSV (comma-separated value) format.	(7)

(*Continued*)

Table 2 Continued

Sequence Step	Description	Working/ Functioning	Next Sequence Step to Follow (Output Which Will be Input to the Further Process)
(7)	Generate SPARQL query to augment new triple pattern (say ? x) like adding a surrogate key [45]	An additional attribute (triple) has been added using the SPARQL query construct clause to generate a new triple (?x) using URI and ROWNUM features. This generates an additional triple with all unique numbers. This has been shown in Figure 5 using the code snippet in Section 5.2.1	(8)
(8)	Produce new RDF dataset (F)	This new RDF data has been generated from sequence step (7)	(10)
(9)	Add triple pattern with SPO on (?x) as a first pattern in SPARQL query (Q')	A new triple pattern with the additional attribute has been added as the first triple pattern of the SPARQL query which has been generated from sequence step (3). This shape of BGP (basic graph pattern) used in the query will help the algorithm to produce better results.	
(10)	Apply Algorithm_Optimizer (F,Q',K,N,P)	Input from sequence step (8) and step (9) along with the value of K, N, and P as the value of subset size, number of triple patterns, and triple pattern used in SPARQL query.	(11)
(11)	Find optimized triple ordering pattern for SPARQL query (Q")	This is the output generated and required to find the optimized triple ordering pattern.	Final outcome (optimized triple pattern order)

The sequence steps (2), (3), (5), (6), (7), (8), and (9) can be omitted if there is already an existence of a prime attribute triple pattern in the RDF dataset or in SPARQL query. In this case, the input of the SPARQL query of sequence step (1) and the RDF dataset in sequence step (4) will directly be passed as an input to Algorithm_Optimizer in sequence step (10).

5.2 Optimization Algorithm Pseudocode

In this section Algorithm 1: Pseudocode is presented of the proposed HSOA, which will give the basic idea of the steps followed to produce the desired results that generate the optimized triple ordering pattern of SPARQL query.

Algorithm 1 Pseudocode for the proposed SPARQL optimization technique (HSOA)

```
Input: SPARQL Query (Q) and RDF data (F)
Output: Optimized SPARQL Query (Q')
Step 1: Analyse the RDF data and identify the possibility of adding information like the
concept of surrogate key in structural data.
Step 2: Introduce the attribute to the existing structure of RDF with the additional subject
?x based on ROWNUM.
Step 3: Create a subset of RDF data (F') based on parameter K (10%) on subject ? x, of
the whole triples existing in the original RDF data.
Step 4: Generate the triple patterns ordering (P) from the SPARQL query (Q)
Step 5: Construct a matrix (M) based on triple pattern ordering on individual pairs of
triple patterns. This matrix contains the time stamp value used with the triple pattern on
the RDF data (F / F') depending on steps 1, 2 & 3.
For i in each triple sequence t
{
    For j in each triple sequence t
     {
        If (i != j)
           {
              Calculate the time required for triple pattern joins (i, j)
              store in time value on matrix index(i,j)
           }
     }
}
Step 6: Construct the shortest path (Path sequence) keeping the source node based on
attribute subject ?x.
Step 7: Generate the reordering of triples based on the path sequence constructed in
step 6.
Step 8: Utilize the reordering triple pattern to construct the final SPARQL query which is
the optimized SPARQL query (Q').
```

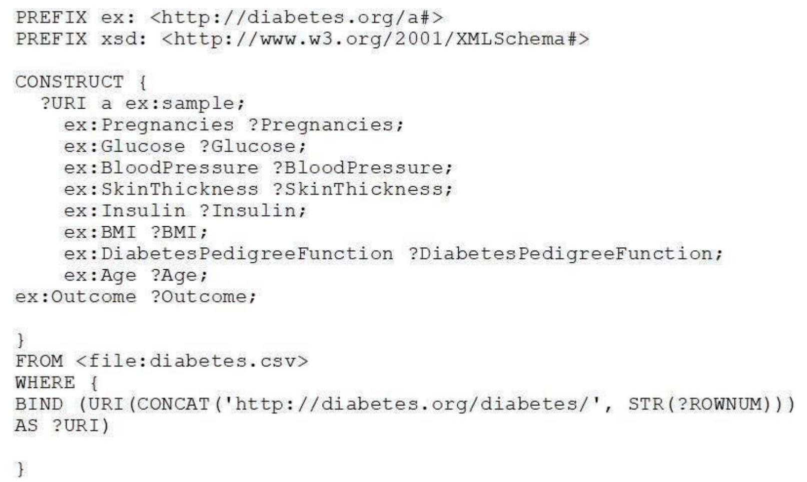
The detailed description of the pseudocode of the algorithm is elaborated in the next sections.

5.2.1 Surrogate key utilization for enhancing the performance for finding the optimized SPARQL query

Based on index-based optimization the surrogate key has been introduced which is an index-based optimization. As SPARQL is based on a joining triple-based pattern, if all the joining is based on the key this will help find faster results for the same. The concept of the surrogate key has been introduced with the existing data. The surrogate key feature has been utilized a lot in data warehousing projects, which results in better performance while joining huge fact tables with wider dimensional tables. A similar utility of a surrogate key has been introduced for existing RDF files or any structured data, to formulate new RDF data having the surrogate key in it [45]. The results of using this concept produce significant results, especially when comparing it with other optimization algorithms for generating optimized SPARQL queries.

This implementation model uses ARQ (an RDF query) engine [43] for information processing and further produces the dataset in comma-separated value (CSV) structured format (with the reference of sequence steps (4), (5) and (6) of Figure 4). Further, SPARQL query has been generated to add a new triple pattern with a key component based on the surrogate key concept with the help of ROWNUM. A sample SPARQL query is shown using a code snippet in Figure 5.

The proposed model produces the new RDF dataset using the above code. Various tools can be used to perform this task like Tarql (transformation SPARQL) [46] which generates RDF datasets. Further, an algorithm optimizer has been used while passing the parameter values of K (an arbitrary number to find subset size say 10% or 20%), N (total number of triples in



```
PREFIX ex: <http://diabetes.org/a#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

CONSTRUCT {
  ?URI a ex:sample;
    ex:Pregnancies ?Pregnancies;
    ex:Glucose ?Glucose;
    ex:BloodPressure ?BloodPressure;
    ex:SkinThickness ?SkinThickness;
    ex:Insulin ?Insulin;
    ex:BMI ?BMI;
    ex:DiabetesPedigreeFunction ?DiabetesPedigreeFunction;
    ex:Age ?Age;
ex:Outcome ?Outcome;

}
FROM <file:diabetes.csv>
WHERE {
BIND (URI(CONCAT('http://diabetes.org/diabetes/', STR(?ROWNUM)))
AS ?URI)

}
```

Figure 5 Code snippet used for generating a triple pattern based on the surrogate key concept using SPARQL and "*Tarql*".

RDF data), and P (number of triple patterns used with SPARQL) discussed in Figure 1 and sequence step (10).

5.2.2 Subset pattern

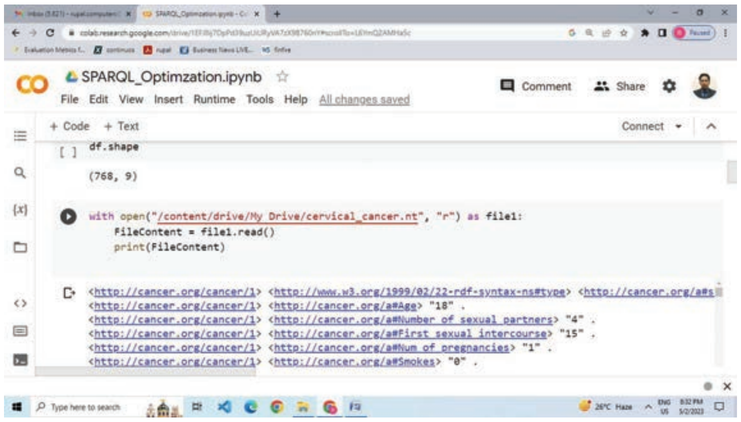
This is another approach, through which the SPARQL optimization can perform even better than the other earlier proposed ways. In the proposed optimization a subset pattern has been used with the surrogate key; the idea behind this approach is that the indexing used by the surrogate key can be further utilized to reduce the steps to be followed. This way will also play a vital role in finding SPARQL optimized query, and it will take less time as compared to previous optimization algorithms because it works on the whole data. These subset patterns will take an initial arbitrary size say K, where K denotes the number of triple patterns like 10% or 20% of the whole data. The SPARQL query works on a Cartesian joining pattern, so with the concept of surrogate key the subset patterns will also work the same while calculating the running time for different triple patterns. This way has also been considered as an efficient way where the optimized SPARQL query can be generated in very little time as compared to previous approaches. The implementation has been done using the concept of file handling in Python. The new RDF dataset has been generated using the search for a specific triple pattern with the index value of K, which is also the size for which the RDF dataset has to be reduced. The Python code snippet has been shown using Figure 6, which uses file handling to read the contents from the original file cervical_cancer.nt and further the new_reduce_file.nt has been generated using the string utility on the Cervical Cancer dataset, where str1 is a temporary string holding the initial RDF data using file content shown in Figure 7.

5.2.3 Generation of a query matrix

This section discusses the query matrix generated based on the triple pattern used in the basic graph pattern of the query. This matrix is generated using the time stamp and the execution time based on the triple patterns joining with the rest of the triple patterns used in the query. This matrix generation is the key concept for calculations which has been used by optimization algorithms to generate optimized SPARQL queries. This intermediate matrix is shown in Figure 8.

5.2.4 Reordering joining patterns based on the shortest path

The joining pattern of the query is the main step for finding the optimized query. The ordering of the SPARQL query has been reordered based on the



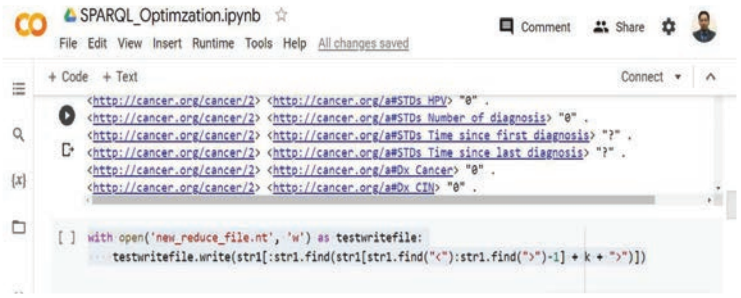
```
df.shape

(768, 9)
```

```
with open("/content/drive/My Drive/cervical_cancer.nt", "r") as file1:
    FileContent = file1.read()
    print(FileContent)
```

```
<http://cancer.org/cancer/1> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://cancer.org/a#s
<http://cancer.org/cancer/1> <http://cancer.org/a#Age> "18" .
<http://cancer.org/cancer/1> <http://cancer.org/a#Number of sexual partners> "4" .
<http://cancer.org/cancer/1> <http://cancer.org/a#First sexual intercourse> "15" .
<http://cancer.org/cancer/1> <http://cancer.org/a#Num of pregnancies> "1" .
<http://cancer.org/cancer/1> <http://cancer.org/a#Smokes> "0" .
```

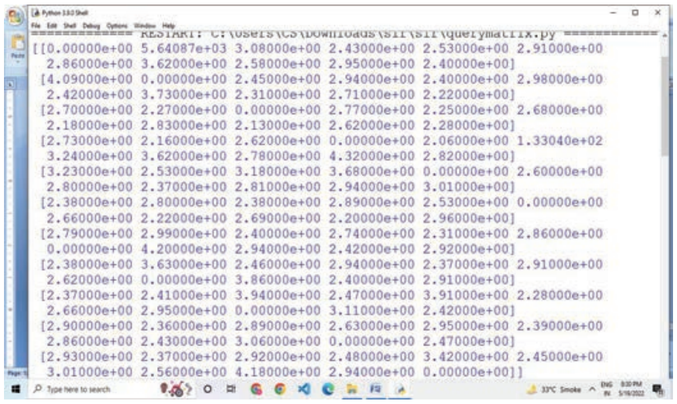
Figure 6 Python code snippet reading the contents of cervical_cancerRDF data.



```
<http://cancer.org/cancer/2> <http://cancer.org/a#STDs HPV> "0" .
<http://cancer.org/cancer/2> <http://cancer.org/a#STDs Number of diagnosis> "0" .
<http://cancer.org/cancer/2> <http://cancer.org/a#STDs Time since first diagnosis> "?" .
<http://cancer.org/cancer/2> <http://cancer.org/a#STDs Time since last diagnosis> "?" .
<http://cancer.org/cancer/2> <http://cancer.org/a#Dx Cancer> "0" .
<http://cancer.org/cancer/2> <http://cancer.org/a#Dx CIN> "0" .
```

```
with open('new_reduce_file.nt', 'w') as testwritefile:
    testwritefile.write(str1[:str1.find(str1[str1.find("<"):str1.find(">")-1] + k + ">")])
```

Figure 7 Python code snippet for generating subset pattern of RDF data on parameter K.



```
============== RESTART: C:\Users\CS\Downloads\SII\SII\querymatrix.py ============
[[0.00000e+00 5.64087e+03 3.08000e+00 2.43000e+00 2.53000e+00 2.91000e+00
  2.86000e+00 3.62000e+00 2.58000e+00 2.95000e+00 2.40000e+00]
 [4.09000e+00 0.00000e+00 2.45000e+00 2.94000e+00 2.40000e+00 2.98000e+00
  2.42000e+00 3.73000e+00 2.31000e+00 2.71000e+00 2.22000e+00]
 [2.70000e+00 2.27000e+00 0.00000e+00 2.77000e+00 2.25000e+00 2.68000e+00
  2.18000e+00 2.83000e+00 2.13000e+00 2.62000e+00 2.28000e+00]
 [2.73000e+00 2.16000e+00 2.62000e+00 0.00000e+00 2.06000e+00 1.33040e+02
  3.24000e+00 3.62000e+00 2.78000e+00 4.32000e+00 2.82000e+00]
 [3.23000e+00 2.53000e+00 3.18000e+00 3.68000e+00 0.00000e+00 2.60000e+00
  2.80000e+00 2.37000e+00 2.81000e+00 2.94000e+00 3.01000e+00]
 [2.38000e+00 2.80000e+00 2.38000e+00 2.89000e+00 2.53000e+00 0.00000e+00
  2.66000e+00 2.22000e+00 2.69000e+00 2.20000e+00 2.96000e+00]
 [2.79000e+00 2.99000e+00 2.40000e+00 2.74000e+00 2.31000e+00 2.86000e+00
  0.00000e+00 4.20000e+00 2.94000e+00 2.42000e+00 2.92000e+00]
 [2.38000e+00 3.63000e+00 2.46000e+00 2.94000e+00 2.37000e+00 2.91000e+00
  2.62000e+00 0.00000e+00 3.86000e+00 2.40000e+00 2.91000e+00]
 [2.37000e+00 2.41000e+00 3.94000e+00 2.47000e+00 3.91000e+00 2.28000e+00
  2.66000e+00 2.95000e+00 0.00000e+00 3.11000e+00 2.42000e+00]
 [2.90000e+00 2.36000e+00 2.89000e+00 2.63000e+00 2.95000e+00 2.39000e+00
  2.86000e+00 2.43000e+00 3.06000e+00 0.00000e+00 2.47000e+00]
 [2.93000e+00 2.37000e+00 2.92000e+00 2.48000e+00 3.42000e+00 2.45000e+00
  3.01000e+00 2.56000e+00 4.18000e+00 2.94000e+00 0.00000e+00]]
```

Figure 8 Snapshot of intermediate matrix generation M (|P|*|P|) string time stamps used for triple pattern joins.

query matrix generated using Section 5.2.3. Further, this has been used with the shortest path/minimum spanning tree algorithm like Prim's to obtain the minimum path and its sequence using the query matrix, which in turn is used to obtain the final optimized reordered joining pattern for the query.

5.2.5 Generate reordering of triple patterns to obtain optimized triple ordering

As per the shortest path calculated from Section 5.2.4, the path originates from the surrogate key concept triple variable (?x). This will help to obtain the triple patterns which are less time-consuming concerning the key triple pattern. This will generate the triple pattern reordering sequence as per the input SPARQL query. This obtained triple pattern sequence will be merged with the existing SPARQL query to obtain the final optimized SPARQL query shown in Table 3.

Table 3 Comparison of SPARQL triple pattern sequence for query (Q1) input query and optimized triple pattern sequence generated using proposed HSOA (algorithm)

Description	Original SPARQL Query Triples	Optimized SPARQL Query Triples
Triple patterns	*0 ?x rdf:typefoaf:Person*	
of the query	*0 ?x rdf:typefoaf:Person*	*10 ?x foaf:Cancer ?Cancer*
	1 ?x foaf:ClumpThickness ?ClumpThickness	*7 ?x foaf:BlandChromatin ?BlandChromatin*
	2 ?x foaf:cell size ?cellsize	*9 ?x foaf:Mitoses ?Mitoses*
	3 ?x foaf:cellshape ?cellshape	*5 ?x foaf:SingleEpithelialCellSize ?singleEpithelialCellSize*
	4 ?x foaf:marginaladhesion ?marginaladhesion	*1 ?x foaf:ClumpThickness ?ClumpThickness*
	5 ?x foaf:SingleEpithelialCellSize ?singleEpithelialCellSize	*3 ?x foaf:cellshape ?cellshape*
	6 ?x foaf:BareNuclei ?BareNuclei	*8 ?x foaf:NormalNucleoli ?NormalNucleoli*
	7 ?x foaf:BlandChromatin ?BlandChromatin	*6 ?x foaf:BareNuclei ?BareNuclei*
	8 ?x foaf:NormalNucleoli ?NormalNucleoli	*2 ?x foaf:cellsize ?cellsize*
	9 ?x foaf:Mitoses ?Mitoses	*4 ?x foaf:marginaladhesion ?marginaladhesion.*
	10 ?x foaf:Cancer ?Cancer	
Pattern sequence results	**Triple pattern sequence of original query *(step 2)* *Algorithm_Optimizer (F,Q,K,N,P)* [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**	**Triple pattern sequence of the optimized query *(step 5)* *Algorithm_Optimizer (F,Q,K,N,P)* [0, 10, 7, 9, 5, 1, 3, 8, 6, 2, 4]**

5.3 SPARQL Optimization Algorithm (HSOA)

Algorithm 2 used for optimization is Algorithm_Optmizer which uses several parameters to work upon. It comprises five subroutines that work sequentially to obtain the outcome as an optimized SPARQL query. These subroutines have been elaborated further in subsequent sections respectively.

Algorithm 2 HSOA (hybrid SPARQL optimization algorithm)

```
Algorithm_Optimizer(F,Q,K,N,P)
Input: (F) is the RDF File in N-Triple Format, (Q) is SPARQL Query, (K) is an arbitrary
    number taking say 10% or 20% value of total size of input File, (N) is the total size of the
    n triples File, (P) is the No. the triple pattern used in SPARQL Query.
Output: Optimized SPARQL Query (Q')
1:  F' <- Construct_SubSet_RDF(F,K,N)
2:  P <- Generate_Pattern_Sequence_SPARQL(Q)
3:  for i in range(P):
4:      for j in range(P):
          M[i][j] <- 0
5:          for i in range(P):
6:              for j in range(P):
7:                  if (i !=j):
          M[i][j] <- Estimate_Execution_Cost(F', Q, i, j)
          // triple i join triple j
8:  Path_Seq<- Shortest_Path(M,|p|,|p|)
9:  Q'<- Generate_Optimize_Query(Q,Path_Seq)
10: return Q'
```

5.3.1 Construct_SubSet_RDF (F, K, N)

This subroutine will generate and return a subset of the dataset (F'). The generation is based on the parameter k which in turn is an arbitrary value, say 10% of the whole dataset triple patterns stored in n-triple format. The existing dataset will work on the dataset using a prime attribute concept in the dataset. The structure of the n-triple pattern will work the same in the SPARQL query processing as per the indexing scheme and triple order pattern evaluation based on join. Considering the above point, it will reduce the time taken for calculating optimized SPARQL query for such type of n-triples RDF data. For the same, Algorithm 3 has been shown as a pseudocode, which used FILE_READ, FILE_WRITE utility of file handling, FIND_Index to find substring index position on file pointer and concatenate and Slice implemented using string handling functions.

Algorithm 3 Pseudocode function Construct_SubSet_RDF (F, K, N)

```
Input: (F) is a RDF dataset File , K is an arbitrary number and (N) is total No. of triples (N)
Output: A RDF subset (F')
1: Identify the URI of the initial triple pattern which has been added using the ROWNUM
   key with ?x source.
2: Data <- FILE_READ(F)
3: X <- Truncate the substring from initial position to beginning of first ?x # identify "<"
   for n-triple data.
4: Y <- generate a value with (%) K on Total triples- N
5: Z <- Truncate the substring after first ?x # identify ">"
6: L <- FIND_Index(Concatenate ( X, Y , Z))
7: SubSet_Data (F') <- FILE_WRITE(Slice[initial index (0) to L])
8: Return (F')
```

For implementation, Python utilities like split, find, list and slicing concepts have been used along with the file handling concepts to generate the subset_rdf dataset. The few key implementations have been shown using the code snippet:

```
str1[:str1.find(str1[str1.find("<"):str1.find(">") -
1] + p + ">")]
```

where N is the count (percentage) of the data in which you want to reduce the dataset. The value of N is based on the size of the dataset that has been read (using the str1 object of the file) and further sliced and written using the file handling object.

5.3.2 Generate_Pattern_Sequence_SPARQL(Q)

This subroutine will generate the triple patterns of the SPARQL query into separate patterns for triples say t0, t1, t2, and so on. This approach will be used to find the optimal triple pattern joining sequence as the path sequence is shown using Path_Seq in step 5. The pattern sequence has been generated using the matrix M (N*M) obtained using algorithm optimizer, where N is the number of triples used for joining patterns in a query. These pattern sequences have been generated directly from the query string which has been passed based on triple patterns. These triple patterns have been stored as a list with the name attributeName shown by below code snippet below. Here str(i) has been used to show pattern_sequence with the triple patterns passed through the query:

```
for i in range(len(attributeName)):
    print(str(i)+attributeName[i])
```

5.3.3 Estimated_Execution_Cost(F',Q,i,j)

This subroutine has been used to find the running time for the SPARQL query on the i$^{\text{th}}$ and j$^{\text{th}}$ triple pattern as a joining sequence. This running time is based on CPU running time taken for the execution of the SPARQL pattern. This has been implemented using the time module utility of Python, where the CPU timestamp has been recorded to calculate the running time taken by the query.

Code snippet:

```
tempq="SELECT * WHERE {{}}"
start=time.perf_counter()
g.query(tempq.format(opt_q_p))
end=time.perf_counter()
print("Time Taken optimized Query")
print(end-start)
```

5.3.4 Shortest_Path(M,|p|,|p|)

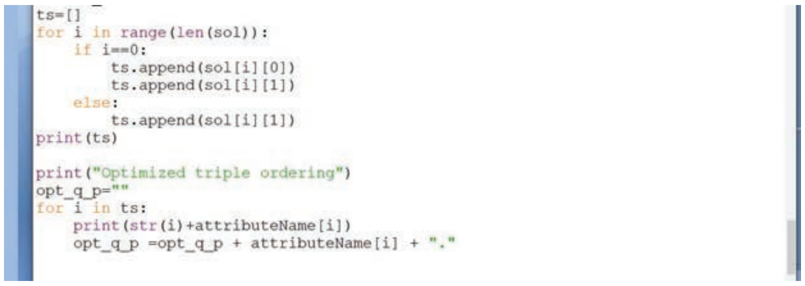
This subroutine will generate the path sequence (reordering of join) based on the triple pattern used in SPARQL query. For an intermediate finding of the final triple pattern, join ordering has been found using Prim's algorithm, which has been ably verified by another shortest path/minimum spanning tree algorithm like Kruskal and Dijkstra. For our experiment, Prim's algorithm has been used to find the optimal triple reordering pattern by fixing the source node ?x as a triple pattern which has been generated with the concept of surrogate key.

5.3.5 Generate_Optimize_Query(Q,Path_Seq)

This subroutine will generate the optimized SPARQL query (Q') which will work efficiently irrespective of the initial given SPARQL query (Q). For generating the optimized path, a minimum spanning tree algorithm has been used to find the shortest path from the source triple covering all the other triple patterns. It has been implemented to optimize the query pattern based on joins.

The path_seq shown in algorithm step 5 has been implemented using ts (triple sequence) and the opt_q_p is the optimized triple pattern sequence generated from path_seq shown in Figure 9.

Further, the optimized path sequence has been used to merge with the query select clause and triple order pattern sequence with the attribute in



```
ts=[]
for i in range(len(sol)):
    if i==0:
        ts.append(sol[i][0])
        ts.append(sol[i][1])
    else:
        ts.append(sol[i][1])
print(ts)

print("Optimized triple ordering")
opt_q_p=""
for i in ts:
    print(str(i)+attributeName[i])
    opt_q_p =opt_q_p + attributeName[i] + "."
```

Figure 9 Intermediate code for generating an optimized path sequence.

terms of triple patterns. The following code has been used to find an optimized SPARQL query:

```
tempq="SELECT ?ClumpThickness ?cellsize ?cellshape
?marginaladhesion ?singleEpithelialCellSize ?BareNuclei
?BlandChromatin ?NormalNucleoli ?Mitoses ?Cancer WHERE {{}}"
Optimized_query=tempq.format(opt_q_p)
```

6 Experimental Setup

For the experiment setup the SPARQL query has been executed and tested on an Intel i3-7100 processor with 2.4 GHz, 8 GB RAM, and 64-bit operating system. Further Python Idle has been used along with the library setup of time, rdflib, numpy, and os modules.

The semantic web has to go a long way regarding the evaluation of systems proposed with semantic web technologies. The semantic web researchers are in the process of creating benchmark datasets for evaluation purposes. Due to the unavailability of benchmark datasets in the field of semantic web technologies, for the evaluation of the proposed system, four benchmark datasets (Cervical Cancer [34], Diabetes [34], Iris [34], and Breast Cancer [34]) generated in RDF n-triple and RDF/XML format from the field of machine learning have been used.

The dataset used in the experimental set has been generated in .nt (N-Triple) RDF format based on datasets available on the UCI repository, which are standard datasets used for various experimental research. The dataset used in RDF forms is based on Cervical Cancer, Diabetes, Iris, and Breast Cancer [34]. Further, for the processing of RDF datasets, SPARQL queries have been developed for information retrieval from the RDF into a desired format which can be further utilized for analysis of its processing.

The evaluation metrics have been used on the time stamps taken from the operating system processor while executing of SPARQL query based on reordering for triple patterns.

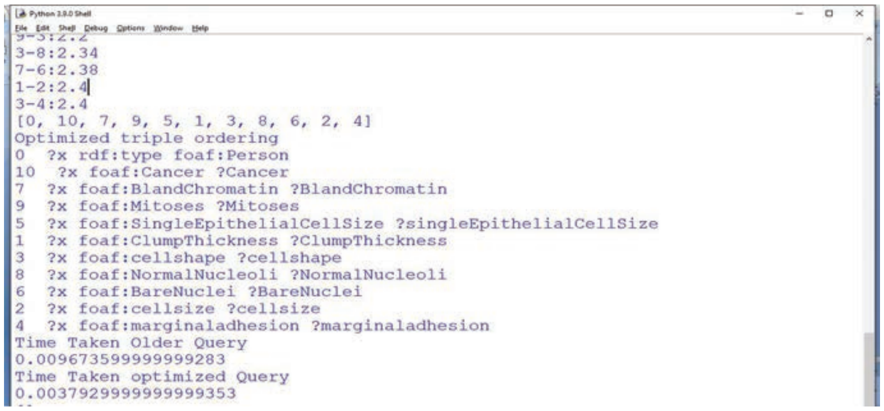
7 Results and Discussions

In this section, the results of SPARQL optimization are observed and analyzed. The information retrieval using SPARQL has been found with different SPARQL queries on different RDF datasets. The comparison of optimized query has been compared with the original query passed on for a specific dataset. Along with the same, the time for calculating optimized SPARQL has also been visualized and compared with the subset patterns using the concept of surrogate key.

7.1 Results of HSOA with Experimental Datasets and Queries Using Python Utilities

Four SPARQL queries (Q1, Q2, Q3, Q4) on different datasets have been used to evaluate the HSOA optimization algorithm. The output produced by the HSOA has been shown using Figures 10, 11, 12, and 13. These elaborate the result of joining patterns for various queries (Q1, Q2, Q3, Q4) on different datasets (Breast Cancer, Diabetes, Iris, and Cervical Cancer) specifications are shown in Table 4.

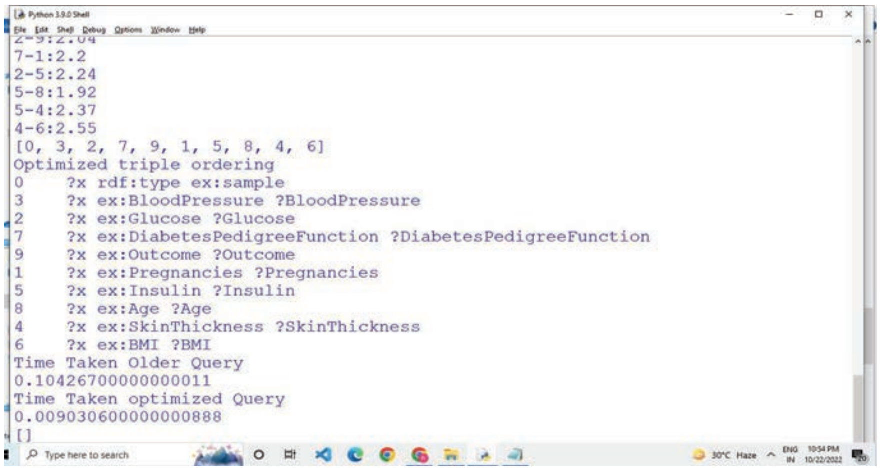
Q1:



```
9-5:2.2
3-8:2.34
7-6:2.38
1-2:2.4
3-4:2.4
[0, 10, 7, 9, 5, 1, 3, 8, 6, 2, 4]
Optimized triple ordering
0  ?x rdf:type foaf:Person
10  ?x foaf:Cancer ?Cancer
7  ?x foaf:BlandChromatin ?BlandChromatin
9  ?x foaf:Mitoses ?Mitoses
5  ?x foaf:SingleEpithelialCellSize ?singleEpithelialCellSize
1  ?x foaf:ClumpThickness ?ClumpThickness
3  ?x foaf:cellshape ?cellshape
8  ?x foaf:NormalNucleoli ?NormalNucleoli
6  ?x foaf:BareNuclei ?BareNuclei
2  ?x foaf:cellsize ?cellsize
4  ?x foaf:marginaladhesion ?marginaladhesion
Time Taken Older Query
0.009673599999999283
Time Taken optimized Query
0.003792999999999353
```

Figure 10 Optimized SPARQL query patterns generated on Breast Cancer RDF data.

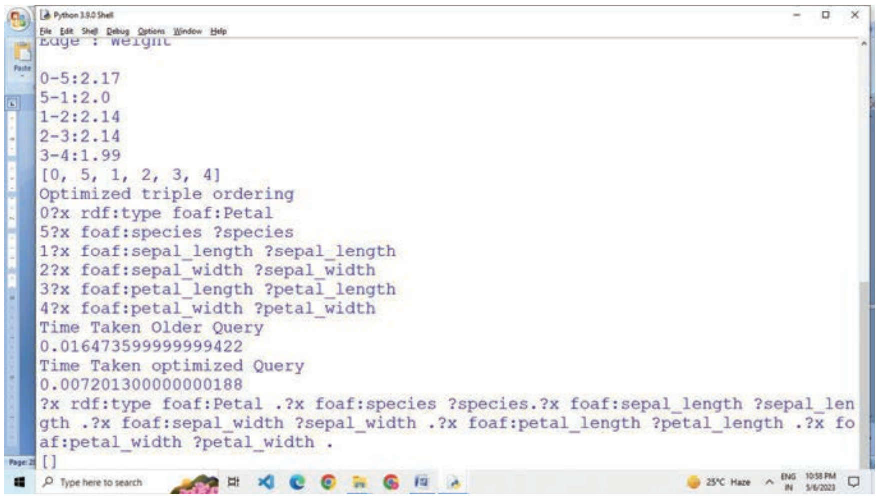
Q2:



```
2-9:2.04
7-1:2.2
2-5:2.24
5-8:1.92
5-4:2.37
4-6:2.55
[0, 3, 2, 7, 9, 1, 5, 8, 4, 6]
Optimized triple ordering
0    ?x rdf:type ex:sample
3    ?x ex:BloodPressure ?BloodPressure
2    ?x ex:Glucose ?Glucose
7    ?x ex:DiabetesPedigreeFunction ?DiabetesPedigreeFunction
9    ?x ex:Outcome ?Outcome
1    ?x ex:Pregnancies ?Pregnancies
5    ?x ex:Insulin ?Insulin
8    ?x ex:Age ?Age
4    ?x ex:SkinThickness ?SkinThickness
6    ?x ex:BMI ?BMI
Time Taken Older Query
0.10426700000000011
Time Taken optimized Query
0.009030600000000888
[]
```

Figure 11 Optimized SPARQL query patterns generated on Diabetes RDF data.

Q3:

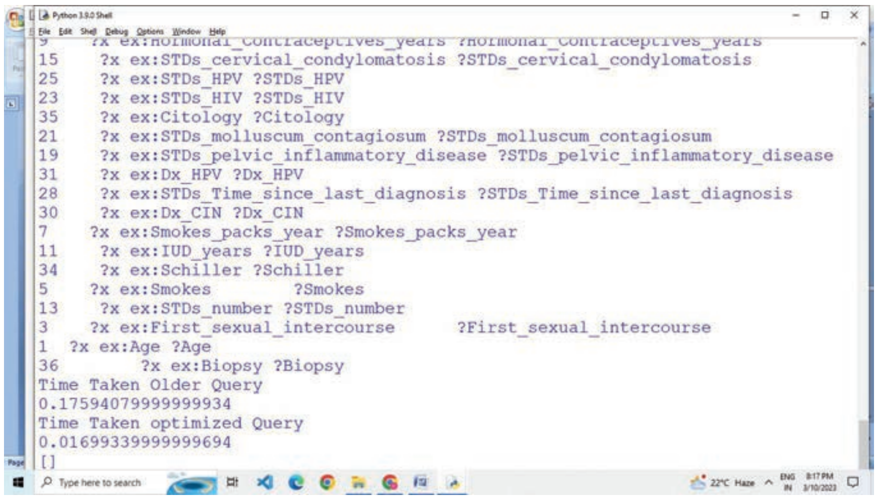


```
Edge : weight

0-5:2.17
5-1:2.0
1-2:2.14
2-3:2.14
3-4:1.99
[0, 5, 1, 2, 3, 4]
Optimized triple ordering
0?x rdf:type foaf:Petal
5?x foaf:species ?species
1?x foaf:sepal_length ?sepal_length
2?x foaf:sepal_width ?sepal_width
3?x foaf:petal_length ?petal_length
4?x foaf:petal_width ?petal_width
Time Taken Older Query
0.016473599999999422
Time Taken optimized Query
0.007201300000000188
?x rdf:type foaf:Petal .?x foaf:species ?species.?x foaf:sepal_length ?sepal_len
gth .?x foaf:sepal_width ?sepal_width .?x foaf:petal_length ?petal_length .?x fo
af:petal_width ?petal_width .
[]
```

Figure 12 Optimized SPARQL query patterns generated on Iris RDF data.

Q4:



```
9      ?x ex:Hormonal_Contraceptives_years ?Hormonal_Contraceptives_years
15      ?x ex:STDs_cervical_condylomatosis ?STDs_cervical_condylomatosis
25      ?x ex:STDs_HPV ?STDs_HPV
23      ?x ex:STDs_HIV ?STDs_HIV
35      ?x ex:Citology ?Citology
21      ?x ex:STDs_molluscum_contagiosum ?STDs_molluscum_contagiosum
19      ?x ex:STDs_pelvic_inflammatory_disease ?STDs_pelvic_inflammatory_disease
31      ?x ex:Dx_HPV ?Dx_HPV
28      ?x ex:STDs_Time_since_last_diagnosis ?STDs_Time_since_last_diagnosis
30      ?x ex:Dx_CIN ?Dx_CIN
7      ?x ex:Smokes_packs_year ?Smokes_packs_year
11      ?x ex:IUD_years ?IUD_years
34      ?x ex:Schiller ?Schiller
5      ?x ex:Smokes        ?Smokes
13      ?x ex:STDs_number ?STDs_number
3      ?x ex:First_sexual_intercourse      ?First_sexual_intercourse
1  ?x ex:Age ?Age
36          ?x ex:Biopsy ?Biopsy
Time Taken Older Query
0.17594079999999934
Time Taken optimized Query
0.01699339999999694
[]
```

Figure 13 Optimized SPARQL query patterns generated on Cervical Cancer RDF data.

It has been found that the results produced by the optimized SPARQL query are identical and perform better than the older queries. The results are presented using a bar graph in Figure 14, and the time taken is shown in seconds.

Table 4 Comparison of running time SPARQL Queries (Q1-Q2-Q3-Q4) with optimized query generated using the proposed HSOA (algorithm)

SPARQL Query	No. Triple Patterns Were Used in Joining Including the Surrogate Key	Dataset	No. of Triples	Time Taken Original Input Query (time in ms)	Time Taken by Optimized Query (Time in ms)
Q1	11	Breast cancer	3149	9.6736	3.793
Q2	10	Diabetes	7680	104.267	9.0306
Q3	6	Iris	1109	16.4736	7.2013
Q4	37	Cervical cancer	31746	175.9408	16.9934

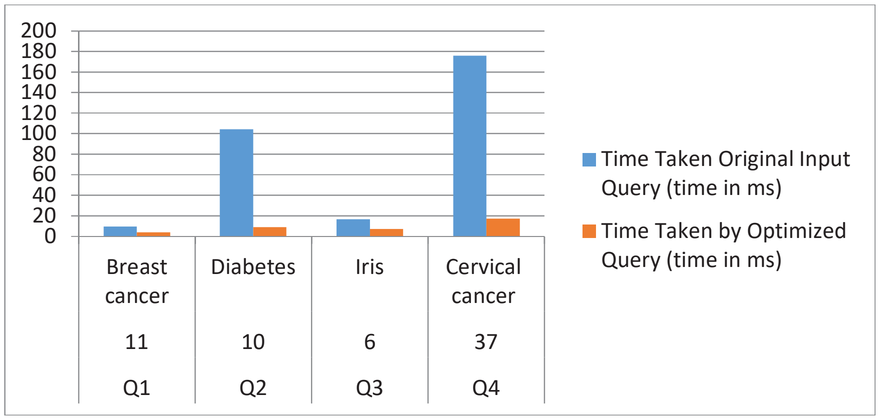


Figure 14 Comparison of running time taken by original and optimized SPARQL query.

7.2 Results of HSOA with Experimental Datasets and Queries on the Stardog Platform

Stardog, a knowledge graph database platform, supports semantic web technologies to store, visualize, and process RDF data using SPARQL [47, 48]. The SPARQL queries used in the paper have also been executed on the Stardog cloud environment by creating an instance of the RDF database and a comparison was made with the optimized queries generated from the proposed algorithm shown in the comparison of running time using Table 5 and Figure 15.

7.3 Comparative Study with Benchmark SPARQL Queries with HSOA

In this section, a comparison has been made of HSOA on the benchmark datasets and queries with LUBM, BSBM [49], and SP2Bench [50]. It was observed that in several benchmark queries, the optimized SPARQL query produced by the HSOA produced better results and is discussed using Tables 6, 7, and 8. The results are shown for all benchmark queries which suited well with the structure of triple patterns used in the experimental setup. Queries taken here use more triple pattern joins and fewer filtering options. The running times shown in Tables 6, 7, and 8 are in milliseconds and represent the AET (average execution time). They can also be visualized using Figures 16, 17, and 18.

Table 5 Comparison of running time of various phases of SPARQL queries (Q1-Q2-Q3-Q4) with optimized query generated using proposed HSOA (algorithm) using the Stardog tool platform

				Time Taken Original Input Query (Time Microseconds)				Time Taken by Optimized Query (Time in Microseconds)			
SPARQL Query	No. of Triple Patterns Used in Joining Including the Surrogate Key	Dataset	No. of Triples	Query Execution Time	Pre-execution Time	Post Execution Time	Total Time	Query Execution Time	Pre-execution Time	Post Execution Time	Total Time
Q1	11	Breast cancer	3149	5	1	0	1372	4	1	0	824
Q2	37	Cervical cancer	31746	71	6	7	2331	39	1	5	999
Q3	6	Iris	1109	3	1	0	2451	1	0	0	2179
Q4	10	Diabetes	7680	13	1	1	2538	9	1	1	1532

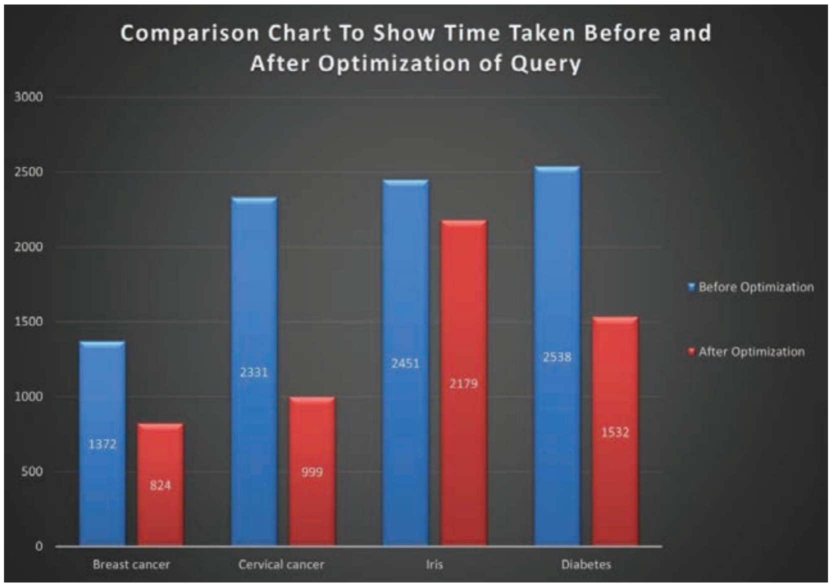


Figure 15 Evaluation of execution time for optimized SPARQL queries using HSOA and using the Stardog tool platform.

Table 6 Comparison of running time in (ms) of BSBM queries (Q1-Q2-Q6-Q8) with the optimized query generated using the proposed algorithm HSOA

BSBM Benchmark	BSBM-50K		BSBM-250K		BSBM-1M	
Query	BSBM-Query	HSOA	BSBM-Query	HSOA	BSBM-Query	HSOA
Q1	7.711	6.654	36.216	33.459	66.769	62.466
Q2	32.37	30.459	46.859	41.898	47.317	43.567
Q6	5.759	5.278	43.215	42.718	49.023	45.5903
Q8	27.343	25.734	60.075	58.935	235.52	224.507

Table 7 Comparison of running time in (ms) of LUBM queries (Q2-Q4-Q8-Q9) with the optimized query generated using the proposed algorithm HSOA

LUBM	LUBM-100		LUBM-1000	
Benchmark Query	LUBM-Query	HSOA	LUBM-Query	HSOA
Q2	2809.2	2579.69	20849.6	19796.508
Q4	218	204.696	272	264.05
Q8	1153.6	1071.968	2564.8	2389.082
Q9	1712.8	1579.308	17383.6	16431.098

Table 8 Comparison of running time in (ms) of SP2Bench queries (Q2-Q4-Q5(b)) with the optimized query generated using the proposed algorithm HSOA

SP2Bench Benchmark	SP2Bench-250K		SP2Bench-5M	
Query	SP2Bench – Query	HSOA	SP2Bench – Query	HSOA
Q2	4.26	4.07256	5.964	5.01402
Q4	3104.262	2822.51412	229509.204	210007.8846
Q5(b)	134.19	128.32185	4432.53	3973.22745

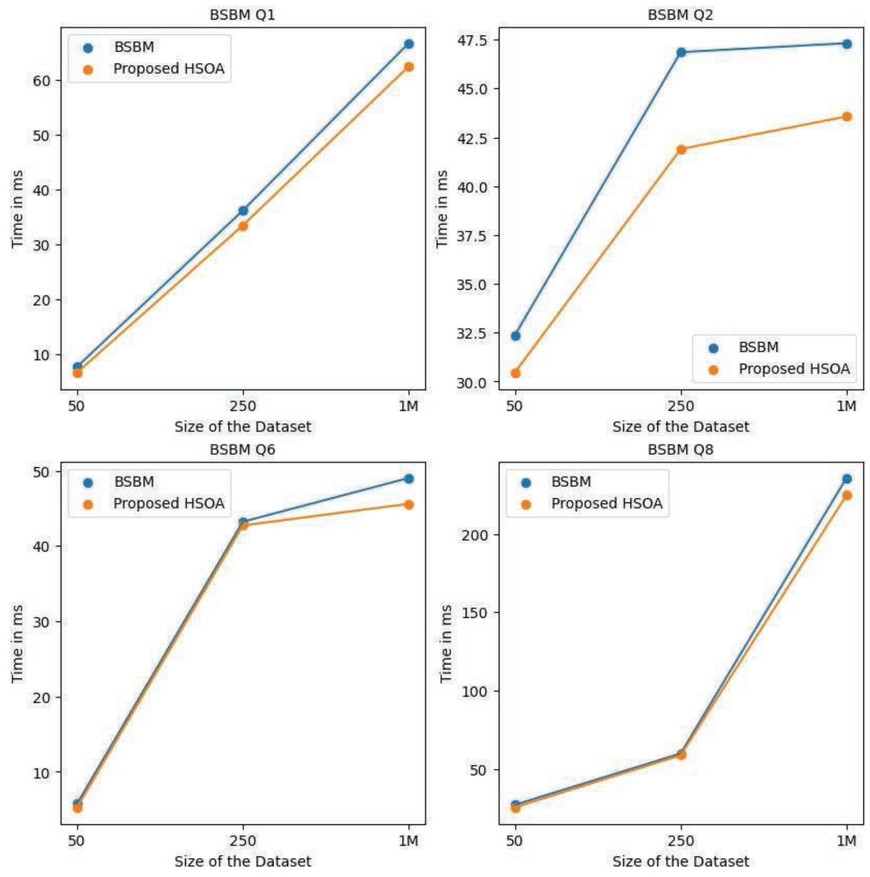


Figure 16 HSOA comparison with the BSBM dataset and queries.

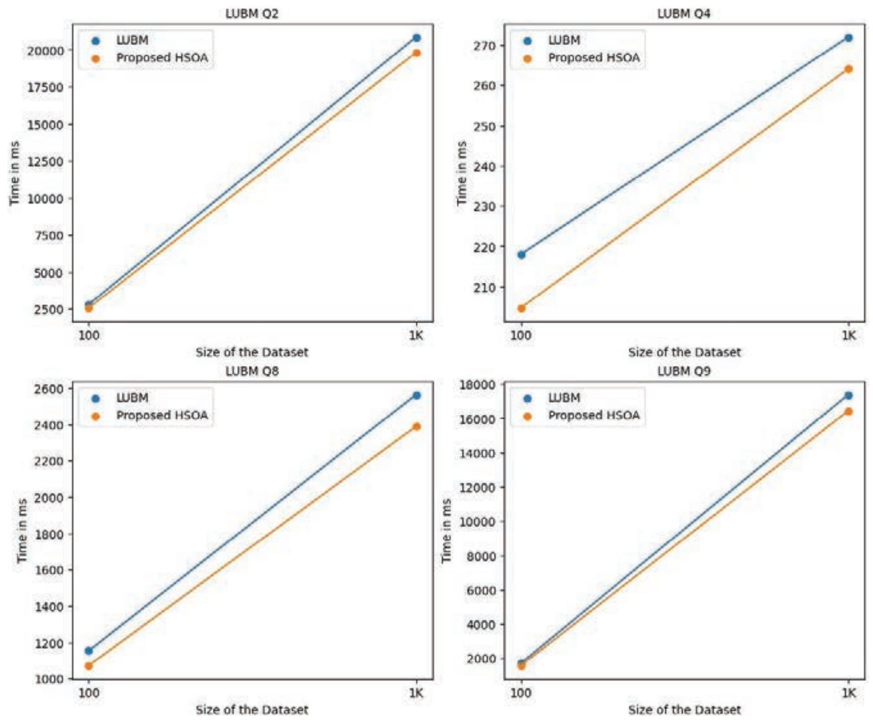


Figure 17 HSOA comparison with the LUBM dataset and queries.

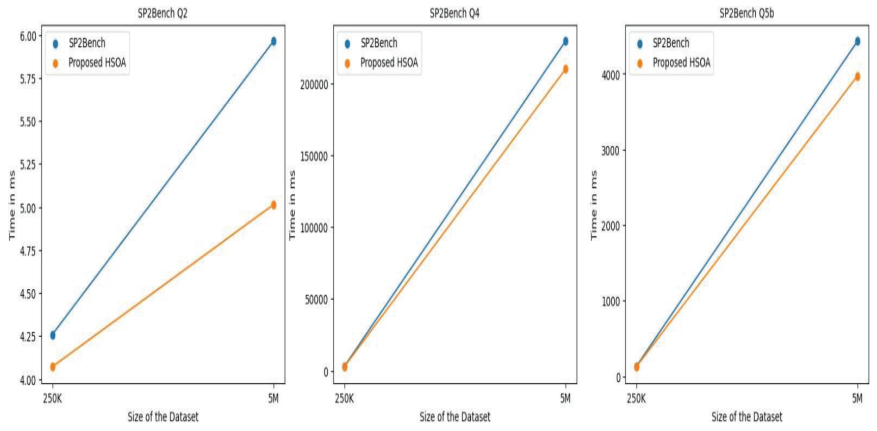


Figure 18 HSOA comparison with the SP2Bench dataset and queries.

8 Conclusions and Future Scope

It has been determined that the SPARQL query is a fundamental element for managing various data kinds on the semantic web for various objectives. For the same, SPARQL uses triple patterns to evaluate the results based on joining patterns. The algorithm HSOA has been proposed for the optimization towards information retrieval from RDF data, which uses the concepts of various optimization techniques that are based on reordering of joining triple patterns based on a surrogate key initially applied on RDF data generation or restructuring the data. Further, the optimization has been also applied to finding the optimized SPARQL query triple pattern on given RDF data. The results produced have also been compared with a few existing versions of SPARQL query on other datasets and it was found that the proposed algorithm produces faster results as compared to a non-optimized way for information retrieval. Four different datasets were used with four respective SPARQL queries. The HSOA algorithm generates optimized SPARQL triple patterns which were effective as compared to the input SPARQL. The results of optimized HSOA were also verified on benchmark datasets. The datasets and SPARQL queries can be utilized by the community for further query processing and machine learning in future aspects.

The HSOA was designed primarily for n-triple RDF datasets and was applied with SPARQL queries to get the optimal execution plan. These SPARQL queries were used to apply machine learning algorithms on RDF datasets. The triple pattern order of execution takes lots of time if it is applied to large RDF datasets. The proposed HSOA algorithm experimental result shows and demonstrates that it is very effective. For other formats of RDF datasets like RDF/XML, turtle and other formats, HSOA can be performed with reordering concepts of optimization. The applicability of surrogate key concepts will be explored for other types of RDF datasets as a future scope.

In the future, a few meta-heuristics algorithms can be explored while integrating their concepts with existing algorithms to enhance the capabilities for finding efficient query plans. Also, the proposed algorithm can be tested to work on real-time applications that use RDF and SPARQL for the semantic web.

References

[1] Guo, J., and Wang, Y.: (2022) RDF Graph Summarization Based on Node Characteristic and Centrality. *Journal of Web Engineering*, pp. 2073–2094.

[2] G. Koutitas, P. Demestichas, (2009) 'A review of energy efficiency in telecommunication networks', Proc. In Telecomm. Forum (TELFOR), pp. 1–4, Serbia, Nov.

[3] Gartner Report, Financial Times, (2007).

[4] I. Cerutti, L. Valcarenghi, P. Castoldi, (2009) 'Designing power-efficient WDM ring networks', ICST Int. Conf. on Networks for Grid Applic., Athens, 2009.

[5] W. Vereecken, et al., (2009) 'Energy Efficiency in thin client solutions', ICST Int. Conf. on Networks for Grid Applic., Athens.

[6] J. Haas, T. Pierce, E. Schutter, (2009) 'Datacenter design guide', whitepaper, the greengrid.

[7] Kalayci, E. G., Kalayci, T. E., and Birant, D. (2015). An ant colony optimization approach for optimizing SPARQL queries by reordering triple patterns. Information Systems, 50, 51–68.

[8] Maillot, P., Corby, O., Faron, C., Gandon, F., and Michel, F. (2023). IndeGx: A model and a framework for indexing RDF knowledge graphs with SPARQL-based test suits. Journal of Web Semantics, 100775.

[9] Ntioudis, D., Masa, P., Karakostas, A., Meditskos, G., Vrochidis, S., and Kompatsiaris, I. (2022). Ontology-Based Personalized Job Recommendation Framework for Migrants and Refugees. Big Data and Cognitive Computing, 6(4), 120.

[10] Guo, Y., Pan, Z., and Heflin, J. (2005). LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics, 3(2–3), 158–182.

[11] S. Groppe, D. Heinrich, C. Blochwitz, T. Pionteck, (2016) "Constructing Large Scale Semantic Web Indices for the Six RDF Collation Orders", Open Journal of Big Data (OJDB), Volume-2, Issue-1, RonPub.

[12] M.D. Nguyen, M.S. Lee, S. Oh and G.C. Fox, (2014) "SPARQL Query Optimization for Structural Indexed RDF Data".

[13] Buwen Wu, Yongluan Zhou, Hai Jin and Amol Deshpande, (2017) "Parallel SPARQL Query Optimization", IEEE 33rd International Conference on Data Engineering (ICDE).

[14] T. Chawla, G. Singh, E.S. Pilli, (2017) "A Shortest Path Approach to SPARQL Chain Query Optimization". International Conference on Advances in Computing, Communications and Informatics (ICACCI).

[15] Papailiou, N., Konstantinou, I., Tsoumakos, D., Karras, P., and Koziris, N. (2013, October). H 2 RDF+: High-performance distributed joins over large-scale RDF graphs. In *2013 IEEE International conference on big data* (pp. 255–263). IEEE.

[16] Hyunsuk Oh, Sejin Chun, Sungkwang Eom, Kyong-Ho Lee, (2015) "Job-Optimized Map-Side Join Processing using MapReduce and HBase with Abstract RDF Data", IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology.

[17] Ge, N., Qin, Z., Peng, P., Li, M., Zou, L., and Li, K. (2022). A cost-driven top-K queries optimization approach on federated RDF systems. *IEEE Transactions on Big Data*.

[18] Li, M., Peng, P., Tian, Z., Qin, Z., Huang, Z., and Liu, Y. (2022). Optimizing Keyword Search Over Federated RDF Systems. *IEEE Transactions on Big Data*.

[19] Peng, P., Ge, Q., Zou, L., Özsu, M. T., Xu, Z., and Zhao, D. (2019). Optimizing multi-query evaluation in federated RDF systems. IEEE Transactions on Knowledge and Data Engineering, 33(4), 1692–1707.

[20] Jose, R. T., and Poulose, S. L. (2019). Semantic Web Query Join Optimization Using Modified Grey Wolf Optimization Algorithm. International Journal of Intelligent Engineering & Systems, 12(5).

[21] Dhiman, G., and Kumar, V. (2018). Emperor penguin optimizer: A bio-inspired algorithm for engineering problems. *Knowledge-Based Systems*, *159*, 20–50.

[22] Dhiman, G., and Kumar, V. (2019). Seagull optimization algorithm: Theory and its applications for large-scale industrial engineering problems. Knowledge-based systems, 165, 169–196.

[23] Dhiman, G., and Kumar, V. (2017). Spotted hyena optimizer: a novel bio-inspired based metaheuristic technique for engineering applications. Advances in Engineering Software, 114, 48–70.

[24] Kaur, S., Awasthi, L. K., Sangal, A. L., and Dhiman, G. (2020). Tunicate Swarm Algorithm: A new bio-inspired based metaheuristic paradigm for global optimization. Engineering Applications of Artificial Intelligence, 90, 103541.

[25] Dhiman, G., Oliva, D., Kaur, A., Singh, K. K., Vimal, S., Sharma, A., and Cengiz, K. (2021). BEPO: A novel binary emperor penguin optimizer for automatic feature selection. Knowledge-Based Systems, 211, 106560.

[26] Dehghani, M., Montazeri, Z., Givi, H., Guerrero, J. M., and Dhiman, G. (2020). Darts game optimizer: A new optimization technique based on darts game. International Journal of Intelligent Engineering and Systems, 13(5), 286–294.

[27] Dehghani, M., Montazeri, Z., Dehghani, A., Ramirez-Mendoza, R. A., Samet, H., Guerrero, J. M., and Dhiman, G. (2020). MLO: Multi Leader

Optimizer. International Journal of Intelligent Engineering & Systems, 13(6).

[28] Dehghani, M., Montazeri, Z., Malik, O. P., Dhiman, G., and Kumar, V. (2019). BOSA: binary orientation search algorithm. International Journal of Innovative Technology and Exploring Engineering, 9(1), 5306–5310.

[29] Dhiman, G., and Kaur, A. (2019). STOA: a bio-inspired based optimization algorithm for industrial engineering problems. Engineering Applications of Artificial Intelligence, 82, 148–174.

[30] Dhiman, G. (2021). ESA: a hybrid bio-inspired metaheuristic optimization approach for engineering problems. Engineering with Computers, 37, 323–353.

[31] Dhiman, G., Garg, M., Nagar, A., Kumar, V., and Dehghani, M. (2021). A novel algorithm for global optimization: rat swarm optimizer. Journal of Ambient Intelligence and Humanized Computing, 12, 8457–8482.

[32] Chawla, T. (2023) "Storage and Query Processing Architectures for RDF Data.", In Encyclopedia of Data Science and Machine Learning (pp. 298–313). IGI Global.

[33] Kelwin Fernandes, Jaime S. Cardoso, and Jessica Fernandes. (2017) 'Transfer Learning with Partial Observability Applied to Cervical Cancer Screening.' Iberian Conference on Pattern Recognition and Image Analysis. Springer International Publishing.

[34] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[35] Thapar, P., and Sharma, L. S. (2022). Implementing SPARQL-based Prefiltering on Jena Fuseki TDB store to reduce the semantic web services search space. In Evolutionary Computing and Mobile Sustainable Networks: Proceedings of ICECMSN 2021 (pp. 319–333). Singapore: Springer Singapore.

[36] Taelman, R., Vander Sande, M., and Verborgh, R. (2019). Bridges between GraphQL and RDF. In W3C Workshop on Web Standardization for Graph Data. W3C.

[37] Leeka, J., &Bedathur, S. (2017). Indexing and query processing in RDF quad-stores (Doctoral dissertation, IIIT-Delhi).

[38] Lin, X., and Jiang, D. (2022). A Two-Phase Method for Optimization of the SPARQL Query. Journal of Sensors, 2022.

[39] Hassan, M., and Bansal, S. (2023). S3QLRDF: distributed SPARQL query processing using Apache Spark—a comparative performance study. Distributed and Parallel Databases, 1–41.

[40] Albahli, S. (2019). Efficient distributed SPARQL queries on Apache Spark. International Journal of Advanced Computer Science and Applications, 10(8).

[41] Ferrada, S., Bustos, B., and Hogan, A. (2022). Similarity Joins and Clustering for SPARQL. Semantic Web Journal IOS press.

[42] Schätzle, A., Przyjaciel-Zablocki, M., Skilevic, S., and Lausen, G. (2015). S2RDF: RDF querying with SPARQL on spark. arXiv preprint arXiv:1512.07021.

[43] Grobe, M. (2009, October). Rdf, jena, sparql and the semantic web'. In Proceedings of the 37th annual ACM SIGUCCS fall conference: communication and collaboration (pp. 131–138).

[44] Lehmann, J., et al. (2015). Dbpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web, 6(2), 167–195.

[45] Palar, P. S., Liem, R. P., Zuhal, L. R., and Shimoyama, K. (2019, July). On the use of surrogate models in engineering design optimization and exploration: The key issues. In Proceedings of the Genetic and Evolutionary Computation Conference Companion (pp. 1592–1602).

[46] Kalampokis, E., Nikolov, A., Haase, P., Cyganiak, R., Stasiewicz, A., Karamanou, A.et,al,. (2014, October). Exploiting Linked Data Cubes with OpenCube Toolkit. In ISWC (Posters & Demos) (pp. 137–140).

[47] Stardog, an enterprise Knowledge Graph platform – https://www.stardo g.com/.

[48] Paradzikovic, P., Hoch, R., and Kaindl, H. (2022). Assigning Systems to Test Environments Through Ontological Reasoning. In Towards a Knowledge-Aware AI (pp. 75–89). IOS Press.

[49] Bizer, C., and Schultz, A. (2009). The Berlin SPARQL benchmark. International Journal on Semantic Web and Information Systems (IJSWIS), 5(2), 1–24.

[50] Schmidt, M., Hornung, T., Lausen, G., and Pinkel, C. (2009, March). SPˆ2Bench: a SPARQL performance benchmark. In 2009 IEEE 25th International Conference on Data Engineering (pp. 222–233). IEEE.

Biographies

Rupal Gupta is a Research Scholar at USIC&T, Guru Gobind Singh Indraprastha University, Delhi, India. His areas of interest are semantic web, SPARQL query processing and optimization, big data, and data mining. He received his Master's, MCA from UPTU, Lucknow, and M.Tech (IT) from USIT, GGSIPU, New Delhi. He has published papers in various conferences and peer-reviewed journals indexed in SCOPUS and Web of Science. He is currently working as an Assistant Professor at Teerthanker Mahaveer University, Moradabad, and has more than 16 years of teaching experience.

Sanjay Kumar Malik completed his Ph.D. in the area of Semantic Web from USIC&T, GGSIP University, Delhi. He is currently working as a Professor in the University School of Information, Communication and Technology, GGSIP University. He has more than 20 years of industry and academic experience in India and abroad (Dubai and USA). His areas of research interest are semantic web and web technologies. He has several research papers published in reputed international conferences (India/abroad) and journals. He has been session chair for several international IEEE/Springer conferences and was honored with the third best researcher award in 2011 by GGSIP University for his research contributions.