A learning approach for query planning on spatio-temporal IoT data

Hoan Nguyen Mau Quoc
Insight Centre for Data Analytics
National University of Ireland Galway, Ireland
hoan.quoc@insight-centre.org

Martin Serrano
Insight Centre for Data Analytics
National University of Ireland Galway, Ireland
martin.serrano@insight-centre.org

John G. Breslin
Insight Centre for Data Analytics and Confirm Centre for Smart Manufacturing
National University of Ireland Galway, Ireland
john.breslin@nuigalway.ie

Danh Le Phuoc
Open Distributed Systems
Technical University of Berlin
danh.lephuoc@tu-berlin.de

ABSTRACT

The ever-increasing growth of the Internet of Things (IoT) has attracted a considerable amount of research attention from the Semantic Web community in order to address the challenge of poor interoperability. However, our survey of research work has shown that the goal of providing an intelligent processing and analysis engine for IoT has still not been fully achieved. Central to this problem is the requirement for a semantic spatio-temporal query processing engine that is able to not only analyze spatio-temporal correlations in a massive amount of IoT data, but that can also generate an effective query plan for a given query to execute in a timely manner. Needless to say, query planning for the multidimensional data like IoT is a costly operation. The most known techniques are either based on the cost model or by using spatio-temporal data statistics and heuristics. In this paper, we propose an alternative solution that uses query similarity identification in conjunction with machine learning techniques to recommend a previously generated query plan to the optimizer for a given query. Our approach also aims to predict the query execution time for the purposes of workload management and capacity planning. Our extensive experiments indicate the efficiency of our learning approach with an impressive prediction accuracy on test queries.

Author Keywords
IoT; Semantic Web; Machine Learning; Query Optimization.

INTRODUCTION

In recent years, we have witnessed a rapid increase in the number of IoT (Internet of Things)-related datasets that are available. To fully realize the potential benefits of these datasets, two fundamental requirements need to be addressed, namely interoperability and effective data management. Fortunately, a suite of technologies developed in the Semantic Web [6] effort, such as the RDF model, Linked Data [7], and SPARQL [23], can be used as some of the principal solutions to help the IoT from the challenge of poor interoperability. However, in this context, providing an effective data management system for IoT that can combine the benefits of Semantic Web principles, and also be able to deal with the "big spatio-temporal data" nature of IoT, is still an open challenge. Central to this problem is not only knowing how to store a massive volume of IoT data, but also being able to answer a complex spatio-temporal related query on large-scale IoT datasets in a timely manner. This requires a query processing engine that has to be "smart" enough to determine an efficient way for executing the given query and that can predict the query behavior in order to take a proper and well-informed decision. Additionally, because spatio-temporal contexts play a vital role in IoT, spatio-temporal computation support is also a mandatory requirement for the engine.

Researchers in the Semantic Web community have proposed a substantial number of works that focus on query optimization in conjunction with supporting spatio-temporal computation. In many of these approaches, for a given query, the query optimizer will firstly analyze the correlation of the spatial, temporal and semantic aspects, and find the best execution plan based on either the complex cost model or by using data statistics and heuristics. However, these approaches have only proven to be effective in a centralized system where the optimizer can easily estimate the cost of all spatio-temporal query execution operations. In the context of a semantic IoT data management system, which requires a distributed integration spatio-temporal query processing engine, these existing query optimization techniques will not be sufficient because of the following reasons: (1) defining a cost model which has to express all spatio-temporal aspects of IoT data and is adaptable to changes in the underlying environment (data communication costs, characteristics of the network, cluster configuration, etc.) is a costly task and has not been fully addressed; (2) the spatio-temporal statistics of IoT datasets are often missing due to the (multidimensional) complexity and high-frequency updates of the data. They are expensive to generate and maintain.

An alternative solution to the above problem is a learning optimizer. The idea of this approach is to reuse existing execution plans to execute similar given queries. In this paper, where we continue our work from [24] and draw upon the ideas in [13], we propose a prediction framework that uses similarity identification between spatio-temporal SPARQL queries as well as machine learning techniques to predict a suitable query execution plan and query execution time for a new query. Specifically, the framework can accurately predict an execution plan that consists of the execution order of all the physical spatio-temporal operators. Moreover, we also aim to predict the query execution time to help the query engine have a better understanding about the query behavior prior to query execution. Accurately predicting the query execution time enables us to optimize our costly cloud infrastructure by rescheduling or preventing long-run queries that might cause too much resource contention or that will not be completed by a particular deadline. In summary, our main contributions are as follows:

1. A framework to predict the best query execution plan for an unforeseen spatio-temporal query
2. A technique to predict query execution times, which enables long-run and resource-consuming queries to be rescheduled or not run at all.

The remainder of the paper is organized as follows. Section 2 briefly introduces spatio-temporal SPARQL queries. In section 3, we review related work on currently existing solutions. Section 4 describes our first approach to modeling and using learning techniques for our spatio-temporal SPARQL query. An experimental evaluation of our proposed learning approach follows in Section 5. A discussion and future work ideas are also given in this section. Finally, we conclude our work in the last section.

PRELIMINARIES

IoT data is always associated with spatio-temporal contexts, thus, the query processing engine for this data needs to support spatio-temporal computation. The current standard query language for RDF, i.e, SPARQL 1.1, does not support spatio-temporal query patterns on semantic data. Recently, there have been several complementary works that aim to support spatio-temporal queries on RDF data. In [22], Perry et al. propose the SPARQL-ST query language, which extends the SPARQL language to support complex spatial and temporal queries on temporal RDF graphs containing spatial objects. Similarly, Koubarakis et al. propose the stSPARQL[15] language which can be used to query stRDF, a data model that is designed for developing ontologies and data instances that encode spatial and temporal information. In the scope of this paper, we will consider the spatio-temporal SPARQL query pattern as following:

```
SELECT [projection clause]
WHERE [spatial pattern]
      [semantic graph pattern]
      [temporal pattern]
ORDER BY [ranking function]
LIMIT [top N results]
```

The **SELECT** clause includes a set of variables that should be instantiated from the RDF knowledge base (variables in SPARQL are denoted by a ? prefix). A graph pattern in the **WHERE** clause consists of triple patterns in the form of <s p o> where any of s, p and o can either be a constant or a variable. The graph pattern is divided into three parts: (1) The spatial pattern consists of a spatial function which is defined by a spatial predicate. The object of the spatial pattern is a set of input variables used for spatial filtering. It should be noted that the spatial functions discussed in our work are equally applicable to all spatial predicates defined in the GeoSPARQL standard by the Open Geospatial Consortium [5]; (2) The semantic graph pattern is the basic RDF triple pattern; (3) The temporal pattern defines temporal filtering through the temporal predicate and time interval variables.

Next, the **ORDER BY** clause helps establish the order of results using a user-defined ranking function. The **LIMIT** clause controls the number of results returned. An exemplary query with a spatio-temporal pattern is:

```
prefix sosa:  <http://www.w3.org/ns/sosa/>
prefix geo:  <http://www.opengis.net/ont/geosparql#>
prefix iot:  <http://iotschema.org/>
prefix temporal:  <http://jena.apache.org/temporal#>
SELECT ?country (?value as ?avg)
WHERE{
?loc geo:winthin(1.675 -47.751 1400 "miles").
?sensor a sosa:Sensor;
        sosa:isHostedBy ?loc;
        sosa:observes iot:AirTemperature.
?loc geo:geometry ?geo;
     geo:parentCountry ?country.
?obs sosa:madeBySensor  ?sensor;
         sosa:hasSimpleResult ?value.
?value temporal:average("01/01/2018" "31/03/2018")
}
GROUP BY (?country)
ORDER BY ?value limit 1
```

Example of spatio-temporal SPARQL query.

This query will detect the coldest European country during the winter of 2018. The spatial patterns of the query will find all the weather stations in Europe through a polygonal range (1.675063 -47.751569 1400 "miles"). The semantic patterns specify the sensor metadata that includes the observation type (iot:AirTemperature). Finally, the time constraint is defined in the temporal patterns with the assumption that the duration of winter in Europe is from January to the end of March.

BACKGROUND AND RELATED WORK

Semantic spatio-temporal query processing and optimization

To process the aforementioned queries, the works in [11, 22, 15] build their own query parser to translate the given spatio-temporal query into an abstract syntax tree. The tree then is mapped to the internal query operators, resulting in a query plan tree. The physical spatio-temporal operators hence is implemented by either extending the Jena/Sesame frameworks or using existing spatial-temporal relational databases. However, these approaches mostly focus on enabling spatio-temporal query features, but hardly any of them fully address the query performance and optimization issues that are associated with querying billions of triples.

As regards dealing with the scalable processing of big spatio-temporal RDF data, distributed solutions are proposed in [16, 29]. These approaches adopt a loosely coupled hybrid architecture, which includes different database systems managed by a central middleware. The middleware works as an integration query processing engine that breaks an input query into sub-queries and delegates them to the proper underlying databases.

For example, the spatial filter of the query is executed by a spatial database while the semantic patterns are processed by an RDF triple store like Jena, Sesame, etc. The query optimizer will then decide a query execution plan that consists of an execution order based on the selectivity estimation of each operator. It is important to note that the execution order has a significant impact on the overall query performance. The first execution operator will prune all unnecessary results, and hence reduce the size of the search space for the next operator. However, generating an optimized execution order only based on a selectivity estimation model without taking into consideration the underlying environment performance - i.e, the data arrival rate, the characteristics of the network, cluster configuration - may face a significant risk that the optimization is critically wrong.

Machine Learning to Predict Query Performance

The recent success of numerous machine-learning-based applications has prompted the database community to investigate the possibilities for integrating machine learning techniques into the design of database systems, especially in query optimizers [1, 19]. This learning technique has become a good alternative to building complex cost models and is often easier to adapt and use with new configurations and data patterns. Moreover, it also enables the prediction of query execution performance which can benefit many system management decisions, including workload management, query scheduling, system sizing and capacity planning. Following this direction, in the relational database area, approaches that exploit machine learning techniques to build predictive models have been proposed [2, 9]. These approaches predict the query execution time and the size of the results by comparing the similarity of query execution plan features. Instead of just predicting query performance, the works in [4, 28, 20] use machine learning to estimate the percentage of work done or to produce an abstract number that is intended to represent a relative query optimizer's cost.

In spite of a sharp growth in the number of such published works, there are still limited works that use machine learning for query optimization on the Semantic Web. Hasan [13] proposed a state-of-the-art approach that uses machine learning to predict SPARQL query execution times. In this approach, Hasan firstly computes the similarity of a SPARQL query based on its algebraic expression and graph pattern, and then predicts the execution time by applying multiple regression SVR. The model is evaluated using DBPSB benchmark queries [21] on an open source triple store (Jena TDB). Following the success of this paper, Zhang et al. [30] present an improvement on this by replacing the clustering model with a hybrid feature vector. It is claimed that the training time is reduced significantly. However, in practice, we observed that applying these approaches for predicting query performance on spatio-temporal IoT data might give inaccurate results. This is because IoT data not only has semantic but also has spatio-temporal aspects, while these approaches only consider the semantic aspect of data in similarity computations. Our work will address this issue by taking into account the spatio-temporal dimensions and query cardinalities in similarity identification.

A LEARNING APPROACH FOR QUERY EXECUTION PLANNING

Overview

In this section, we describe our proposed learning approach for query planning on spatio-temporal IoT data. Our approach is illustrated in Figure 1.

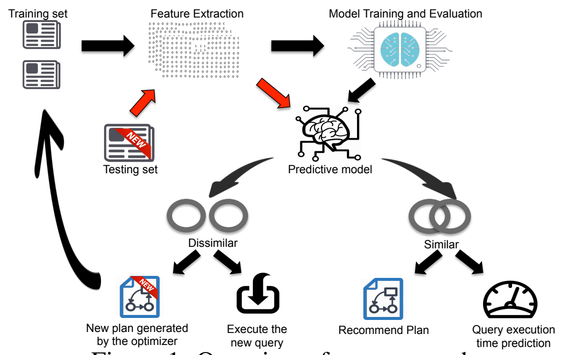
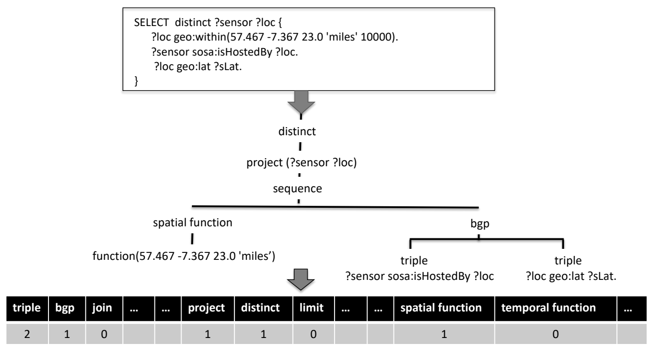


Figure 1: Overview of our approach

The approach consists of two distinct phases: an offline training phase and an online testing phase. Before starting the training phase, we firstly select the historical queries set and split it into three subsets for training, validation and testing. These query sets will be executed on the target system from which the query execution plan and execution time are observed. To execute the query, we build an offline automatic script that runs separately to the training phase. The script is used to detect the optimized execution plan for each given query by considering all the possible plans. The plan which has the lowest cost will be selected. In the next step, the Features Extractor module will extract all the selected query features into the features matrix, which is a necessary input for machine learning algorithms. The features selection process will be presented in the following section.

The next component is Model Training and Evaluation, which is responsible for evaluating the efficiency of different learning models. For that, this component will train the models with the training queries set and then evaluate them by using the validation set. During the training and evaluation processes, the setting of parameters for each model can be optimized by using cross-validation procedures, i.e, the number of clusters, the k value for k-NN regression models, etc. The testing set, which is not used during model training, can be used for final testing and model comparison. By evaluating on a testing set, the learning model with the most accurate prediction will be selected as a Predictive Model, which will be used in the testing phase.

The second phase of our approach is online testing that runs the predictive model with a real-world query. For a given query Q_n, the Predictive Model will propose a suitable execution plan, already generated during the training phase, and reuse it to execute Q_n. For that, the Predictive Model needs to compute the similarity of Q_n with trained queries that were previously executed by the optimizer. If similarity is detected between Q_n and a trained query Q_t , then the execution plan P_t used by the optimizer to execute Q_t will be reused to process Q_n. The same methodology is applied for query execution time prediction.



```
SELECT  distinct ?sensor ?loc {
     ?loc geo:within(57.467 -7.367 23.0 'miles' 10000).
     ?sensor sosa:isHostedBy ?loc.
      ?loc geo:lat ?sLat.
}
```

triple	bgp	join	...	...	project	distinct	limit	...	...	spatial function	temporal function	...
2	1	0			1	1	0			1	0	

Figure 2: Algebra Feature Extractor

Query Features Extraction Process

A typical setup for machine learning is to be given a collection of vectors containing all of the feature values for a given data set. Therefore, we represent our spatio-temporal query into its feature vector representation. In this study, we build upon the work in [13] which uses algebra and graph pattern features to build the features vector. The difference here is that we extend the SPARQL algebraic expression vector by adding spatial and temporal metrics. For the graph pattern, we not only extract the SPARQL basic graph pattern, but also consider the spatial and temporal patterns in our graph similarity computation. In addition, as well as the algebra and graph patterns, we also include query cardinality features. According to our experimental results, this extra feature proves its effectiveness by improving the prediction results significantly.

SPARQL algebra features with a spatio-temporal extension

This step is used to transform the spatio-temporal SPARQL query strings to algebraic expressions. For an input query, the query engine will parse the query to an algebraic expression tree. In this tree, non-leaf nodes are algebraic operators such as joins, and leaf nodes are the variables present in the triple patterns of the initial query. Additionally, we add two more operators that are unique in our engine: the spatial and temporal operators. After parsing a query string to an algebraic expression, our Feature Extractor component will calculate the frequencies of all the algebraic operators as query features. Figure 2 shows an example of extracting the query algebra features vector from a spatio-temporal SPARQL query.

As shown later in our experiments, the prediction obtained using a similarity computation based on query algebra features is quite accurate in recognizing the queries that are almost identical. However, by carrying out a further analysis into the incorrectly predicted results, we recognized two significant risks:

1. The predictive model may judge that two queries are not similar when analyzing their algebraic expression trees, even though their execution plans are actually very similar.

2. The predictive model may judge that two queries have a high similarity measure when comparing their algebraic expressions, even though their execution plans are different. This is due to the fact that even small differences in query graph patterns or differences in the cardinality of each query operator, which would normally be ignored in the textual presentation of an algebraic expression, can be very decisive during the process of defining an execution plan.

To address this problem, we take into account the query graph pattern similarity, which will be described in the next section.

Spatio-temporal graph patterns features

In respect of query planning, judging two queries as being similar based on their algebraic expressions can present a significant risk in terms of incorrect prediction. We realized when computing query similarities based on algebraic features that we only considered the frequency of the query operator, i.e, the number of triples in the basic graph patterns appearing the queries, but failed to represent the query graph structure. Recalling that a SPARQL query can also be considered as an RDF graph, it is therefore obvious that queries that have some structural similarity might potentially share the same execution plan and query performance.

To represent the spatio-temporal graph patterns, we propose building graph pattern features. Specifically, we transform the similarity problem of two query patterns to the similarity problem of two graphs. To compute the structural similarity between two query patterns, we first construct two graphs from the two query patterns, then compute the graph edit distance (**GED**) between these two graphs. The GED between two graphs is the minimum amount of distortion needed to transform one graph to another [8]. The process to extract the graph patterns features can be briefly described as follows:

1. We cluster the structurally similar graph patterns in the training data into K_{med} clusters. We apply the same clustering algorithms proposed in [13], which are the K-mediods clustering algorithm [14] and Risen's graph distance [26], to cluster the training queries. However, in addition to the standard SPARQL graph patterns, we also consider the spatio-temporal graph patterns when computing the graph similarity. We built our own graph pattern extractor to not only extract the semantic graph pattern but also to extract the spatial and temporal patterns. The spatio-temporal graph pattern will thus be considered as a standard RDF graph in which the edges of the graph are spatio-temporal functions and the nodes of the graph are the input variables for the functions. We call such a graph a spatio-temporal graph. Figure 3 illustrates the graph representation of our sample spatio-temporal query. To ensure that the spatio-temporal triples are more important than standard triples, we heuristically improve the weighting function in the GED method by increasing the cost of edit operations on the spatio-temporal graph.

2. After having created the set of K_{med} clusters in Step 1, we then compute the GED between the graph of each query q_i in the training set and the central graph pattern in each cluster, and regard each distance as an instance of a feature. As a result, we obtain a K_{med}-dimensional feature vector for q_i, where K_{med} is the number of clusters.

Query cardinality features

We build query cardinality features by selecting the cardinality of each dimension of the input query. We observe that there are some queries that share structurally similar basic graph patterns and the same algebraic expressions, but neither the query execution plan nor the query execution time are similar. This is because the cardinalities (spatial cardinality, temporal cardinality, semantic cardinality) are not considered in the similarity computation for both the algebra and graph pattern

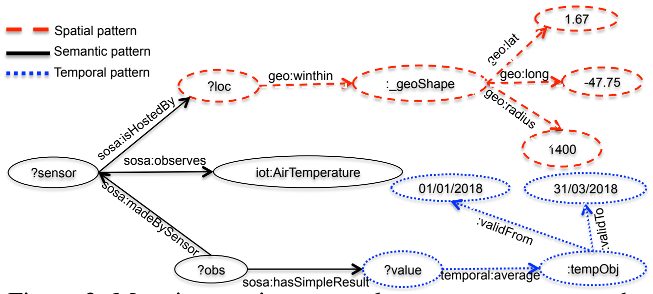


Figure 3: Mapping spatio-temporal query patterns to graphs

features. Note that the cardinalities will help the optimizer to decide the execution order of each operator. For example, if the spatial cardinality of the query is small, the query plan should process the spatial operator first, pruning unnecessary results, followed by the semantic part of the query. For simplicity, we assume that the temporal operator always has the highest cardinality due to the massive amount of historical IoT data that can be available. Hence, we limit our study to selecting the spatial and semantic cardinalities in order to create a Query Cardinality features vector. We retrieve spatial cardinality quickly by performing the bounding box query defined in the spatial operator. We then reuse the work in [27] to estimate the semantic cardinality. A brief description of these features is given in Table 1.

Feature	**Description**
Sp_ARD	The estimated result size of spatial operators
Se_ARD	The estimated result size of semantic patterns

Table 1: Description of the Query Cardinality Features

EXPERIMENTS

Experimental Setup

Setup and software

The backend system for our query engine [24] is running on a physical cluster of four servers. We train the model on a separate server that has the following configuration: 2x E5-2609 V2 Intel Quad-Core 2.5GHz 10MB Cache, Hard Drive 3x 2TB 7200RPM, Memory 32GB1600MHz DDR3.

We experimented with Jena v2.13.1 to extract the algebra features and have used Weka v3.8.2 [12] to implement the machine learning algorithms. The GED was calculated using the Graph Matching Toolkit [25].

Dataset

We use a weather dataset, which is a subset of our **Graph of Things** (GoT) dataset [17] for training and testing our learning model. To generate the queries set, we collect the GoT query logs that have been gathered since 2016 and classify them into 13 different query templates based on their characteristics (query operators, temporal range, observed properties, etc.). The query templates analysis summary is described in Table 2. Each template consists of placeholders that can be replaced by specific terms. The temporal, semantic and spatial metrics indicate the computational complexity of the query template. From these templates, the training, validation and test queries will then be generated. To generate queries, we randomly assign selected RDF terms from the GoT dataset to the placeholders in the query templates. We generate over 2244 queries that are divided equally between all templates. We take about 40% of the queries for training purposes, about 27% for validation and about 33% for testing. This process results in 900 training queries, 600 validation queries and 744 testing queries.

Template	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Number of placeholder	6	4	6	5	3	5	9	6	7	5	7	8	6
Number of triple	5	3	6	5	1	3	9	9	4	9	11	12	12
Temporal	✗	✗	✗	✗	✓	✓	✓	✗	✗	✗	✓	✓	✓
Semantic	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✓	✓
Spatial	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓

Table 2: Query templates analysis

Predictive models

In this paper, we evaluate three machine learning algorithms available in the Weka library. Each algorithm is considered as the most representative machine learning algorithm in its category. We firstly run our experiment with the Support Vector Machine for regression (SVR) [10]. The SVR creates a maximum-margin hyperplane to split the input features to a higher dimensional space and then performs a regression in that space via a linear regression function. We performed the SVR test with two commonly used kernels: the RBF and Polynomial kernels. The second algorithm is the k-Nearest Neighbors (k-NN) [3]. Conceptually, k-NN is a lazy learning algorithm that predicts based on the closet training data point. For our experiment, we apply two variations of k-NN by considering different distance functions: Euclidean distance and Manhattan distance. The last learning model in our experiment is Random Forest [18], which is a typical representation of a Decision Trees Regression model. Random Forest is an improvement upon bagged decision trees that disrupts the greedy splitting algorithm during tree creation so that split points can only be selected from a random subset of the input attributes.

Evaluation Metrics

Our work applies classification and regression models to query plans and query execution times, respectively. Therefore, we use a classification metric, called *accuracy*, to evaluate the prediction of query execution plans. For query execution time, we use *root mean square error* and the *coefficient of determination* as our prediction metrics. Details of each of these metrics are described as follows:

- Accuracy: The accuracy metric measures the ratio of correct predictions to the total number of instances evaluated. The prediction accuracy is defined as:

$$Accuracy = \frac{1}{n}\sum_{i=1}^{n} 1(\hat{y}_i = y_i) \tag{1}$$

  where $\hat{y}_i$ is the predicted value and y_i is the corresponding actual value, with n being the total number of observations.

- Root Mean Square Error (RMSE): RMSE is used to measure the differences between values predicted by a model and the actual values. The RMSE is defined as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}} \tag{2}$$

- Coefficient of Determination (R^2): The R^2 value provides a measure of how well future samples are likely to be predicted by the model. The R^2 value is always between 0 and

1. An R^2 score close to 1 indicates near perfect prediction. R^2 is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{\sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2}, \text{ where } \bar{y} \text{ is the mean of the actual value} \quad (3)$$

Experimental Results

Experiment 1: Train a model with algebraic features

In this experiment, we evaluate the performance of different learning algorithms, namely SVR-RBF and SRV-Polynomial (SVR-Pol), k-NN with Euclidean distance (KNN-Eu), k-NN with Manhattan distance (KNN-Mah), and Random Forest (RF). We firstly compare the prediction accuracy of these algorithms for the query execution plan and then present the results of their performance for query execution time. For the k-NN algorithms, we selected the k value, the number of neighbors, using cross validation. We found that k=2 gives us the best results.

Table 3 reports the execution plan prediction accuracy. We observe that the accuracy values of these algorithms are just slightly different, rounding to 80%. The highest value is realized by the RF algorithm, followed by SVR-Pol and KNN-Eu.

	SVR-RBF	SVR-Pol	KNN-Eu	KNN-Mah	RF
Accuracy	79.704%	80.108%	79.973%	79.963%	80.127%

Table 3: Query plan prediction accuracy using Algebra features

In the following, we look at the query execution time prediction results. The regression performances of the learning algorithms are evaluated by RMSE and R^2, as shown in Table 4. One interesting fact we noticed is that, in contrast to the execution plan prediction, the SVR algorithms (SVR-Pol and SVR-RBF) have poor performance. The minimal values of R^2 (0.01 and 0.113) are given by the SVR algorithms while the k-NNs and RF perform in almost the same way with values equal to 0.976 and 0.975, respectively. The R^2 values indicate that the k-NN and RF algorithms predict more accurately than the SVR algorithms. Additionally, the RMSE values also imply bad performance of the SVR algorithms with high values of 366.359 and 357.288 for SVR-RBF and SVR-Pol. To plot a comparison of the predicted and actual execution time values, we select the algorithms that have the best performance in each group, which are SVR-Pol, KNN-Eu and RF. The plots are shown in Figure 4. We use a log-scale to accommodate the wide range of query execution times. As illustrated in these figures, there are still a substantial number of outliers, which indicate incorrect prediction results. According to our observations, these incorrect predictions fall into the category of queries which share the same algebraic expression but have different query plans and execution times.

	SVR-RBF	SVR-Pol	KNN-Eu (k=2)	KNN-Mah (k=2)	RF
RMSE	366.359	357.288	54.809	154.245	54.927
R^2	0.01	0.113	0.976	0.976	0.975

Table 4: R^2 and RMSE values using Algebra features

Experiment 2: Train a model with algebraic features and spatio-temporal graph patterns

In the second experiment, we again use only three algorithms which are derived from the previous experiment, namely SVR with Polynomial kernel, k-NN with Euclidean distance and Random Forest. Using the cross-validation procedure, we are able to select and set the number of clusters for the k-mediods clustering algorithm at K_{med} = 25, and the number of neighbors for k-NN at k=2. We chose these values because they give us the lowest RMSE and highest R^2 values. Table 5 presents the query plan prediction accuracy. By adding the spatio-temporal graph pattern features in our similarity computation, we achieve the highest accuracy value of 87.68% with the RF algorithm. Following this is the KNN-Eu one with 86.42%. The SVR-Pol has increased slightly at 80.64%.

Figure 5 and Table 6 portray the prediction results for query execution time. As shown in Figure 5, the prediction results of KNN-Eu and RF move closer to a perfect prediction. The highest R^2 value is 0.979 (KNN-Eu). Analyzing the results, we realized that by adding the graph patterns features, the number of inaccurate predictions for two queries that have structurally similar spatio-temporal graph patterns but different algebraic expressions is reduced significantly. This achievement is also explained by the decrease in RMSE values as described in Table 6. The most considerable decrease in RMSE belongs to KNN-Eu, which is 54.809. The RMSE of the SVR-Pol algorithm also substantially reduces, which is now 129.23. These are the main improvements from the first experiment with the algebraic features model.

	SVR-Pol	KNN-Eu (k=2)	RF
Accuracy	80.645%	86.425%	87.688%

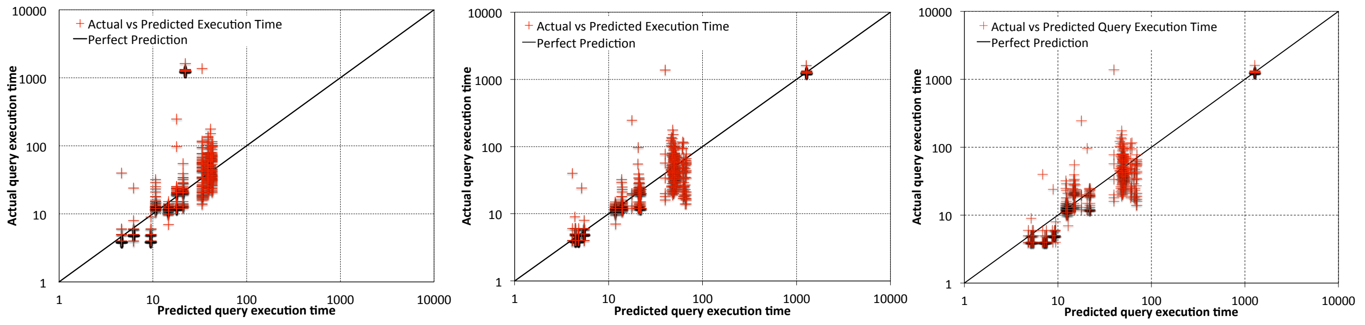
Table 5: Query plan prediction accuracy using Algebra and Graph patterns features

	SVR-Pol	KNN-Eu (k=2)	RF
RMSE	129.231	54.809	54.927
R^2	0.970	0.979	0.976

Table 6: R^2 and RMSE values using Algebra and Graph patterns features

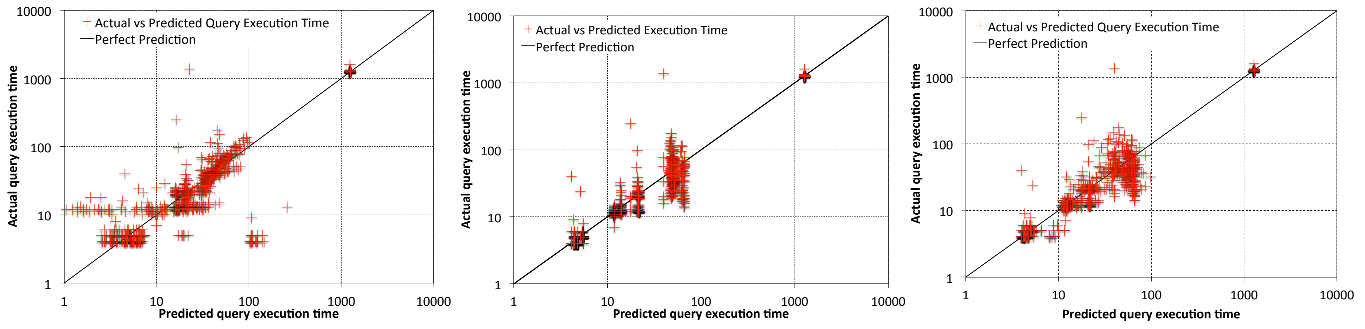
Experiment 3: Train a model with algebraic features, spatio-temporal graph patterns and query cardinality features

So far, we have described the experimental results for predicting query execution plans and query execution times using algebra and graph patterns as feature vectors. We also ran the experiment by adding query cardinality features. As mentioned in Section 4, this additional feature vector helps to improve the predictive accuracy of spatio-temporal queries that are similar in terms of algebraic expression and graph pattern representation, but that differ in query performance. Not surprisingly, the prediction results of this experiment are more accurate compared to Experiment 2. This is evidenced by the increased prediction accuracy values shown in Table 7, and the fewer outliers with respect to the perfect prediction line in Figure 6. Specifically, in Table 7, KNN and RF continually show their efficiency with the highest achieved accuracies of 95.054% and 95.860% with regard to query execution plan prediction. For the query execution time prediction results, as described in Table 8, we get an overall best R^2 value of 0.990



(a) SVR with Polynomial kernel (b) k-NN with Euclidean distance (c) Random Forest

Figure 4: Regression-predicted vs. actual execution times using Algebra features



(a) SVR with Polynomial kernel (b) k-NN with Euclidean distance (c) Random Forest

Figure 5: Regression-predicted vs. actual execution times using Algebra and Graph patterns features

(using RF) and an overall lowest RMSE value of 35.746 (using RF). These results indicate that our model can accurately predict the query plan and execution time for a given query.

	SVR-Pol	KNN-Eu (k=2)	RF
Accuracy	93.414%	95.054%	95.860%

Table 7: Query plan prediction accuracy using Algebra, Graph patterns and Query cardinality

	SVR-Pol	KNN-Eu (k=2)	RF
RMSE	61.062	39.273	35.746
R^2	0.972	0.987	0.990

Table 8: R^2 and RMSE values using Algebra, Graph patterns and Query cardinality

Discussion and future work

Our extensive experimental results demonstrate the efficiency of our learning approach for query planning on spatio-temporal IoT data. The most significant improvement in our approach is achieved by the inclusion of spatio-temporal information in the query similarity computation. Also, together with the algebraic expression and spatio-temporal graph patterns, we propose a new feature vector, namely query cardinality, to enhance the prediction accuracy in the case of queries that share similar algebraic expressions and graph patterns, but that have different query performance. The experimental results indicate that our approach can accurately predict over 95% of input queries.

For future work, we want to address some of the drawbacks that exist in our study. One major limitation of our approach is its poor adaptability to dynamic streaming IoT data. We are investigating techniques to improve the learning model so as to be amenable to continuous retraining in the context of streaming queries. Another drawback is that temporal cardinality is not adjusted in our learning model. This is due to our implicit assumption that this cardinality always has the largest value compared to the other cardinalities. We have a work-in-progress to efficiently estimate the temporal cardinality, and hence, this issue will be resolved in the near future.

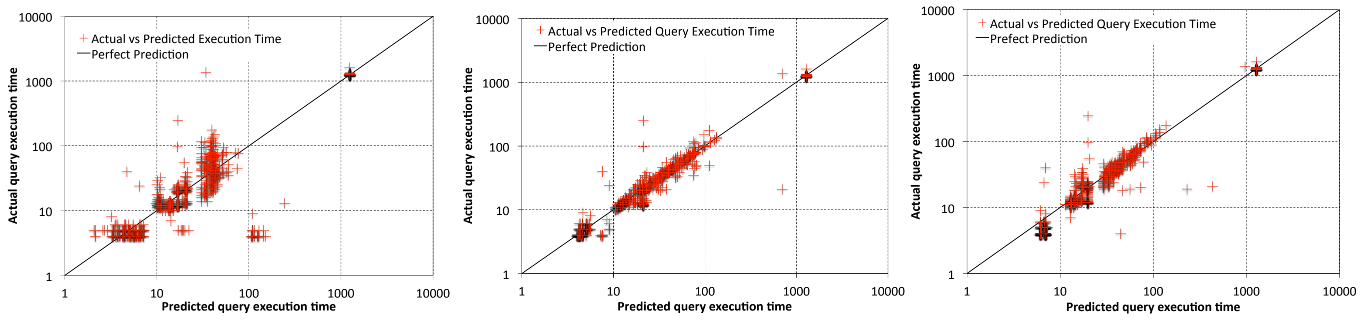
CONCLUSION

In this paper, we present our recent efforts to accurately predict both a query execution plan and the query execution time using similarity identification techniques on spatio-temporal SPARQL queries. Our goal was to build a machine learning model that recognizes the similarity of queries, and only using the information that is available before the queries are to be executed. Compared to existing works, the major contribution of our approach is the use of machine learning to improve similarity detection by not only taking into account semantic aspects but also spatio-temporal correlations. Whilst our approach still has its limitations, it is a step towards providing an alternative solution that aims to address the query optimization challenge for the spatio-temporal management of IoT data.

REFERENCES

1. Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Trans. on Knowl. and Data Eng.* 17, 6 (June 2005).

2. Mert Akdere and others. 2012. Learning-based Query Performance Modeling and Prediction. In *Proceedings of the 2012 IEEE ICDE*.



(a) SVR-Polynomial kernel (b) k-NN with Euclidean distance (c) Random Forest

Figure 6: Regression-predicted vs. actual execution times using Algebra, Graph pattern and Query cardinality features

3. Naomi S Altman. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician* 46 (1992).
4. Martin Arlitt. 2000. Characterizing web user sessions. *ACM SIGMETRICS Performance Evaluation Review* 28, 2 (2000), 50–63.
5. Robert Battle and Dave Kolas. 2012. Enabling the geospatial semantic web with parliament and geosparql. *Semantic Web* 3 (2012).
6. Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The semantic web. *Scientific american* (2001).
7. Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. 2008. Linked data on the web (LDOW2008). In *Proceedings of the 17th World Wide Web*. ACM.
8. Horst Bunke and Gudrun Allermann. 1983. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters* 1 (1983).
9. Archana Ganapathi and others. 2009. Predicting Multiple Metrics for Queries: Better Decisions Enabled by Machine Learning. In *Proceedings of the 2009 ICDE*. IEEE Computer Society, 592–603.
10. Steve R Gunn and others. 1998. Support vector machines for classification and regression. *ISIS report* (1998).
11. Claudio Gutierrez, Carlos A. Hurtado, and Alejandro Vaisman. 2007. Introducing Time into RDF. *IEEE Trans. on Knowl. and Data Eng.* 19 (February 2007). Issue 2.
12. Mark Hall and others. 2009. The WEKA data mining software: an update. *ACM SIGKDD* 11 (2009).
13. Rakebul Hasan and Fabien Gandon. 2014. A Machine Learning Approach to SPARQL Query Performance Prediction. In *Proceedings of the 2014 IEEE/WIC/ACM WI-IAT - Volume 01*. IEEE Computer Society, 266–273.
14. Leonard Kaufman and Peter Rousseeuw. 1987. *Clustering by means of medoids*. North-Holland.
15. M. Koubarakis and K. Kyzirakos. 2010. Modeling and querying metadata in the semantic sensor web: the model strdf and the query language stsparql. In *Proc. ESWC*, Vol. 12. 425–439.
16. Kostis Kyzirakos and others. 2012. Strabon: a semantic geospatial DBMS. In *ISWC*. Springer.
17. Danh Le-Phuoc and others. 2016. The Graph of Things: A step towards the Live Knowledge Graph of connected things. *Journal of Web Semantics* 37 (2016).
18. Andy Liaw and others. 2002. Classification and regression by Random Forest. *R news* 2, 3 (2002).
19. Patrick Marcel and Elsa Negre. 2011. A survey of query recommendation techniques for data warehouse exploration.. In *EDA*. 119–134.
20. Justin Moore and others. 2005. Data center workload monitoring, analysis, and emulation. In *8th CAECW*. 1–8.
21. Mohamed Morsey and others. 2011. DBpedia SPARQL benchmark-performance assessment with real queries on real data. In *ISWC*. Springer, 454–469.
22. M. Perry and others. 2011. Sparql-st: extending sparql to support spatiotemporal queries. *Journal of Geospatial Semantics and the Semantic Web* 12 (2011).
23. Eric Prud, Andy Seaborne, and others. 2006. SPARQL query language for RDF. (2006).
24. Hoan Nguyen Mau Quoc and Danh Le Phuoc. 2015. An Elastic and Scalable Spatiotemporal Query Processing for Linked Sensor Data. In *11th SEMANTICS*. NY, USA.
25. Kaspar Riesen and others. 2013. A novel software toolkit for graph edit distance computation. In *GbRPR*. Springer.
26. Kaspar Riesen and Horst Bunke. 2009. Approximate graph edit distance computation by means of bipartite graph matching. *Image and Vision computing* 27 (2009).
27. Markus Stocker and others. 2008. SPARQL basic graph pattern optimization using selectivity estimation. In *WWW '08*. 595–604.
28. Richard M Yoo and others. 2006. Constructing a non-linear model with neural networks for workload characterization. In *Workload Characterization, 2006 IEEE International Symposium on*. IEEE, 150–159.
29. Liang Yu, Yong Liu, and Jong Lee. 2012. SSTDE: an open source semantic spatiotemporal data engine for sensor web. In *ACM SIGSPATIAL Workshop*. ACM.
30. Wei Zhang and others. 2016. Learning-based SPARQL Query Performance Prediction. In *WISE*. Springer.