Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

Predicting an Optimal Virtual Data Model for Uniform Access to Large Heterogeneous Data

CHAHRAZED B.BACHIR BELMEHDI, ABDERRAHMANE KHIAT AND NABIL KESKES
LabRI-SBA, Enterprise Information Systems
ESI-SBA Institute; Fraunhofer IAIS
Algeria; Germany
E-mail: {cb.bachirbelmehdi;n.keskes}@esi-sba.dz; abderrahmane.khiat@iais.fraunhofer.de

The growth of generated data in the industry requires new efficient big data integration approaches for uniform data access by end-users to perform better business operations. Data virtualization systems, including Ontology-Based Data Access (ODBA) query data on-the-fly against the original data sources without any prior data materialization. Existing approaches by design use a fixed model e.g., TABULAR as the only *Virtual Data Model* - a uniform schema built on-the-fly to load, transform, and join relevant data. While other data models, such as GRAPH or DOCUMENT, are more flexible and, thus, can be more suitable for some common types of queries, such as join or nested queries. Those queries are hard to predict because they depend on many criteria, such as query plan, data model, data size, and operations. To address the problem of selecting the optimal virtual data model for queries on large datasets, we present a new approach that (1) builds on the principal of OBDA to query and join large heterogeneous data in a distributed manner and (2) calls a deep learning method to predict the optimal virtual data model using features extracted from SPARQL queries. OPTIMA - implementation of our approach currently leverages state-of-the-art Big Data technologies, Apache-Spark and Graphx, and implements two virtual data models, GRAPH and TABULAR, and supports out-of-the-box five data sources models: property graph, document-based, e.g., wide-columnar, relational, and tabular, stored in Neo4j, MongoDB, Cassandra, MySQL, and CSV respectively. Extensive experiments show that our approach is returning the optimal virtual model with an accuracy of 0.831, thus, a reduction in query execution time of over 40% for the tabular model selection and over 30% for the graph model selection.

Keywords: Data Virtualization, Big Data, OBDA, Deep Learning.

1. INTRODUCTION

Massive data generated by applications, transactions, or machines keep increasing drastically over the years [1]. However, the information extracted from this data is unexploited and less used, leading to a knowledge gap [2]. Consequently, the growing volume of data consumed by different applications raises the need for effective data integration approaches [3, 4]. The aim is to get more insights by enabling the process of a large volume of data that is stored in various sources (Oracle, MongoDB, etc.), that is resided in different platforms (cloud, mainframes), and is represented in different formats (relational, graph, no-relational [5]). Modern approaches "Data virtualization [6]"

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

tackle this challenge by creating a virtual data model under which the heterogeneous formats are homogenized *on-the-fly* without data materialization [7], thus reducing cost, and simplifying data management, updates, and maintenance. Ontology-based data access (OBDA) [8] also implements a virtual data model and addressed data integration challenges with practical knowledge representation models, ontology-based mappings, and a unique query language SPARQL[1] [9].

Existing approaches [10, 11, 12] use by design only one virtual data model[2] (e.g., TABULAR) to load and transform the requested data into a uniform model to be joined and aggregated; while other data models, such as GRAPH or DOCUMENT, are more suitable [13]. For instance, approaches using a fixed TABULAR virtual model (TABULAR is a model that uses predefined structures, i.e., table definitions) can have downside performances for SPARQL queries that involve many join operations on very large data. In contrast, other data models such as GRAPH (a model that structures data into a set of nodes, relationships, properties, and, most importantly, stores relationships at the individual record level) perform better for such queries. On the other hand, the TABULAR model performs better for queries that involve selection or projection. The problem to be addressed in this paper is defined as, *given a query, "which virtual data model is optimal i.e., the model that has the lowest query execution time (cost)? and how to select it?"*.

It is very challenging, however, to automatically select the optimal virtual model based on queries since it is not realistic to compute the query execution time for all SPARQL queries against all virtual data models to get the actual cost. Furthermore, the query behavior on data virtualization is quite hard to predict since the behavior depends not only on the virtual data model but also on query planning. To the best of our knowledge, existing machine learning techniques [14, 15, 16] [17] were established in the literature for cost estimation of SPARQL queries; most of them, however, are designed for querying uniform data, e.g., RDF[3] and not for distributed data sources.

To address these research questions, we developed OPTIMA - an OBDA extensible framework that predicts the optimal virtual data model GRAPH or TABULAR, using a deep learning algorithm to join data from sources databases that support Property Graph, Relational, Tabular Document-based, and Wide-Columnar models. The proposed algorithm uses one hot vector encoding to transform different SPARQL features into hidden representations. Next, it embeds these representations into a tree-structured model, which is used to classify the virtual model GRAPH or TABULAR that has the lowest query execution time.

Extensive experiments show that our approach is successfully running, returning the optimal virtual model with an accuracy of 0.831, thus reducing the query execution time of over 40% for the TABULAR model selection and over 30% for the GRAPH model selection.

The article is structured as follows. The underlying concepts about ontology-based big data access are given in Section 2. Our approach is described in detail in Section 3. Further description of deep learning model is presented in Section 4. Experimental results are reported and explained in Section 5. Related Work is presented in Section 6. Section 7

[1] SPARQL is a query language for Resource Description Framework (RDF).

[2] We denote GRAPH and TABULAR when referring to the type of virtual data model; while we denote Property Graph, Document-based, Wide-Columnar, Relational, and Tabular when addressing the source model.

[3] Resource Description Framework (RDF) is a standard designed as a data model for describing metadata.

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

concludes with an outlook on possible future work.

2. Preliminaries

Our proposed approach requires the following inputs (1) data sources using different models. (2) Semantic Mapping that describes mapping in RDF Mapping Language, (3) information about data sources (password, etc.), and (4) a set of SPARQL queries. To guide the subsequent description of our approach, we provide the following definitions:

Definition 1 (Data Source Schema) *Dataset Schema is a set of $S_d \cup S_c \cup S_r \cup S_g \cup S_t$ considered by our approach; we introduce each model briefly as follows:*

- *Document-based S_d [18]: A document d is a JSON object o. An object is formed by a set of key/value pairs (aka fields) $o = \{ k_1 \dots k_n \}$; a key is a string, while a value can be either a primitive value (e.g., a string), an array of values, an object, or null.*

- *Wide-Columnar S_c [19]: A table t is the unit of wide-column identified by name and composed by a set of column-families f. The table's rows are identified by a unique key. Each row of the table can contain up to n records. The record is a pair of identifiers id and a value. A wide-column is, in fact, a Hash structure expressed as: $t = Hashtable < key, Hashrow < f, Hashrecord < id, value >>>$.*

- *Relational S_r [20]: A relation schema R with a set Σ of PKs, FKs and attributes $A =< A_1, \dots, A_n >$ is denoted $R(A_1, \dots, A_n)$ is a set of $n-tuples < d_1, ..., d_n >$ where each di is an element of $dom(Ai)$ or is null. The relation instance is the extension of the relation. A value of null represents a missing or unknown value.*

- *Property Graph S_g [21] $G = (V, E, \lambda, \mu)$ is a directed, edge-labeled, attributed multi-graph where V is a set of nodes, $E \subseteq (VxV)$ is a set of directed edges, $\lambda : E \to \Sigma$ is an edge labeling function assigning a label from the alphabet Σ to each edge. Properties can be assigned to edges and nodes by the function $\mu : (V \cup E)xK \to S$ where K is a set of property keys and S the set of property values.*

- *Tabular S_t [22] is a set of tables $T = \{ t_1 \dots t_n \}$. Each table t_x integrates one or more column groups, as $t_x = \{ GC_1 \dots GC_n \}$. Each column group integrates different columns representing the atomic values to be stored in the table, $GC_x = \{ C_1^x \dots C_n^x \}$.*

We denote an entity of a data source by $e_x^s = \{a_i\}$, representing either a node, a table or an object; where s is the schema entity, x its name and a_i^x are its attributes representing either edges or columns. A data source consists of one or more entities, $d = \{e_i\}$.

Definition 2 (Semantic Mapping) *Semantic mappings are bridges (links) between the ontology and sources schemata elements. We differentiate between two types of semantic mappings [12]:*

- *Entity mapping: $m^{en} = (e, c)$ a relation mapping an entity e from d onto an ontology class c.*

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

- *Attribute mapping:* $m^{at}=(a,p)$ *a relation mapping an attribute a from an entity e onto an ontology property p.*

Definition 3 (Star-Shaped Query) *A Star-Shaped Query (SSQ) is a set of triples (subject, predicate, object) patterns - BGPs*[4] *sharing the same subject [23]. We denote SSQ by* $st_x = \{t_i = (x, p_i, o_i) \mid t \in BGP_q\}$ *where x is the shared subject, whereas* $BGP_q = \{(s_i, p_i, o_i) \mid p_i \in O\}$*, is the triple patterns of SSQ.*

Definition 4 (Connection SSQ) *The joins of data coming from different data sources are represented actually by the connections between star-shaped queries i.e., two SSQs* st_a*,* st_b *(subject, predicate, object) are connected if the object of* st_a *is the subject of* st_b*.* $connected(st_a, st_b) \rightarrow \exists t_i = (s_i, p_i, b) \in st_a$.

Definition 5 (Relevant Entities to SSQ) *[24] An entity e is relevant to a SSQ st if it contains attributes* a_i *mapping to every triple property* p_i *of the SSQ i.e.,* $relevant(e,st) \rightarrow \forall p_i \in prop(st) \exists a_j \in e \mid (p_i, a_j) \in M^{at}$*, where prop is a relation returning the set of properties of a given SSQ.*

Definition 6 (Entity Wrapping) *it is a function wrap that takes one or more relevant entities to SSQ and returns a Virtual Model [24]. It loads entity elements and organizes them according to Virtual model schema* $wrap : E^n \rightarrow PS$.

Definition 7 (Virtual Data Model) *Virtual Data Model is the data structure of the computation unit of the query engine to load, transform and join only the relevant data. It is built and populated on-the-fly and not materialized, i.e., used only during query processing then cleared. Virtual Data Model has a schema that organizes data according to its structure. We consider two types of schema, GRAPH or TABULAR.*

- *Structure of a GRAPH [25] (in-memory) is similar as Property Graph. A GRAPH* $G = (V,E)$ *is a set of vertices* $V = \{1 \ldots n\}$ *and a set of m directed edges E. The directed edge* $(i, j) \in E$ *connects the source vertex* $i \in V$ *with the target vertex* $i \in V$*. GRAPH stores relationships at the individual record level.*
- *Structure of a TABULAR (in-memory) [26] is the same structure as the Tabular model defined above. TABULAR has predefined structures.*

Definition 8 (Graph and Data Parallel) *During the querying execution, the Virtual Model, GRAPH or TABULAR is partitioned, distributed, and queried in parallel.*

- *GRAPH Parallel*[5] *is executed after loading relevant entities into the DEE. Graph-Parallel Systems consist of a property graph G = (V, E, P) and a vertex-program Q that is instantiated simultaneously on all the vertices.*
- *Data Parallel [27] concerns the TABULAR model, which is executed after loading relevant entities into the DEE. Data-Parallel computation derives parallelism by processing independent data on separate resources.*

[4]Basic Graph Pattern (BGP) is a set of Triple Patterns, where BGPs is set of BGP.
[5]https://gist.github.com/shagunsodhani/c72bc1928aeef40280c9

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

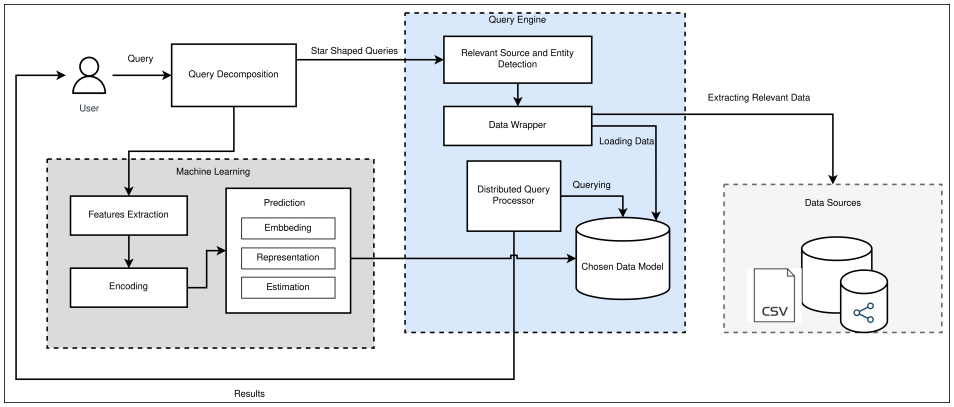


Fig. 1: Predicting Optimal Virtual Model on top of OBDA

3. Predicting Optimal Virtual Model for Querying Large Heterogeneous Data

To solve the problem of selecting the optimal virtual data model and thus efficiently query large heterogeneous data, we propose an approach that leverages OBDA methodology and deep learning. Our Solution follows OBDA and supports two types of virtual data models, GRAPH and TABULAR, to load and join data from sources with various models, i.e., property graph, document-based, wide-columnar, relational, and tabular. We used a deep learning algorithm that predicts the optimal virtual model based on query behavior. More precisely, the algorithm extracts and encodes significant features from input SPARQL query into representations that are then embedded into a tree-structured model to classify the virtual model, GRAPH or TABULAR, that has the lowest cost i.e., query execution time. Below we describe each part of our proposed approach illustrated in Figure 1.

3.1 Virtual Data Model Prediction

Our distinctive deep learning model, built on top of OBDA layers, aims to select the optimal virtual data model based on query behavior. Our algorithm analyzes and extracts features from the input SPARQL query and uses One-Hot Vector encoding[6] to transform different features into hidden representations. Next, these representations are embedded into a tree-structured model, which can effectively learn the representations of query plan features and predicts the cost against each virtual data model. As an output, the proposed algorithm returns the optimal virtual model, GRAPH or TABULAR, that has the lowest query execution time. Our deep learning algorithm is detailed in section 4. Once the optimal model is predicted, the rest of the OBDA layers (e.g., query decomposition, entity detection, and operations, e.g., join, limit) follow the optimal virtual data model, GRAPH or TABULAR.

[6]One-hot vector is a $1 \times N$ matrix (vector) used to distinguish each word in a vocabulary from every other word in the vocabulary.

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

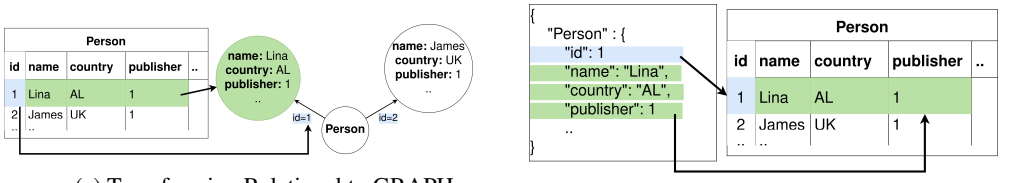
3.2 Query Decomposition & Relevant Entity Detection

Once the optimal virtual model is selected, our approach decomposes the input SPARQL query into star-shaped queries to identify conjunctive queries [28]. More precisely, in SPARQL, the conjunction is expressed using shared variables across sets of triple patterns, also called basic graph patterns (BGP). Based on this characterization, we divide the query's BGP into a set of sub-BGPs, where each sub-BGP contains all the triple patterns sharing the same subject variable - called star-shaped query - SSQ (Definition 3). Most approaches for query decomposition in OBDA systems follow subject-based method because triples sharing the same subject correspond to the same entity, e.g., table or object in the data source, thus avoiding traversing data to find specific entities to be joined and extra joins that can be very expensive.

Next, our approach analyzes each star-shaped query and retrieves semantic mappings that are already predefined i.e., correspondences between SSQ elements/variables (i.e., ontology class or property) and data sources' entities (e.g., table) or attributes (e.g., column name) in addition to data source type (e.g., relational) [see Definition 2]. A correspondence that maps every triple property of a star-shaped query is called a relevant entity (Definition 5). Finally, loading those entities defined by data sources' models into the optimal virtual data model, GRAPH or TABULAR, requires data mapping and transformation, for instance, mapping and transforming a table from a relational model into a GRAPH or TABULAR. Furthermore, star-shaped SPARQL operations (e.g., Projection, filtering, grouping, etc.) are also translated into GRAPH or TABULAR operations.

3.3 Data Mapping and Transformation

Once the relevant entities and sources are identified using semantic mappings as shown above, our approach maps and transforms relevant entities (e.g., a table) from their original models (e.g., relational) [Definition 1] to data that comply with optimal virtual data model predicted, GRAPH or TABULAR (Definition 7). This conversion occurs at query-time, which allows for the parallel execution of expensive operations, e.g., join (Definition 6).



(a) Transforming Relational to GRAPH

(b) Transforming Document-based to TABULAR

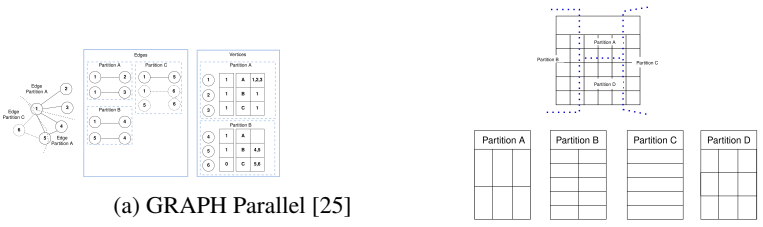
Fig. 2: Transformation Process

Each star-shaped query corresponds to one relevant entity, and thus one single virtual data model is created. This is the case when the relevant entity, according to the mapping, could be retrieved only from one data source, e.g., one relational table. Otherwise, if the relevant entity according to the mapping could be retrieved from multiple sources, then the virtual model for the entity is the union of temporary virtual models created for each source (Figure 4).

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

Below we describe data source models transformation by wrappers into GRAPH and TABULAR.

- For the virtual data model of type GRAPH, the structure returned of relevant data on different data sources using existing data access methods [24] is schema-less data, e.g., RDD (Resilient Distributed Dataset). Then necessary structural adaptations are employed, which consist of converting schema-less to GRAPH following the mapping process. The data is represented as a table with specific columns for the Tabular and Relational models defined by CSV and MySQL. Then the mapping process is defined as follows (see Figure 2a): for each table row, a vertex is created with the same label as the table's name (e.g., table 'Person' corresponds to all vertices with the label 'Person') in addition to the root vertex. Edges are created between vertices and the root vertex, whereas the properties of each vertex are the columns of the table (e.g., column 'name' corresponds to property 'name'), and the values of the properties are the table's cell information. The same process is applied to property graphs defined by neo4j, document-based, and Wide-Column models (e.g., an XML file) defined by MongoDB and Cassandra.
- As for the virtual data model of type TABULAR, the structure returned of relevant data on different data sources using existing data access methods is organized into named columns, e.g., DataFrame. Adaptations are needed, which consist of converting DataFrame to TABULAR following a mapping process. For instance, the selected object as a relevant entity of documented-based and wide-columnar stored in MongoDB and Cassandra is parsed to create a virtual TABULAR (see Figure 2a), which consists of a table with a name similar to the root object's name (e.g., a table 'Person' from object name 'Person'). A new row is inserted by iterating through object elements into the corresponding table. The corresponding key-values are saved under the column representing the cell information. The same process is applied to other models.



(a) GRAPH Parallel [25]

(b) TABULAR Parallel

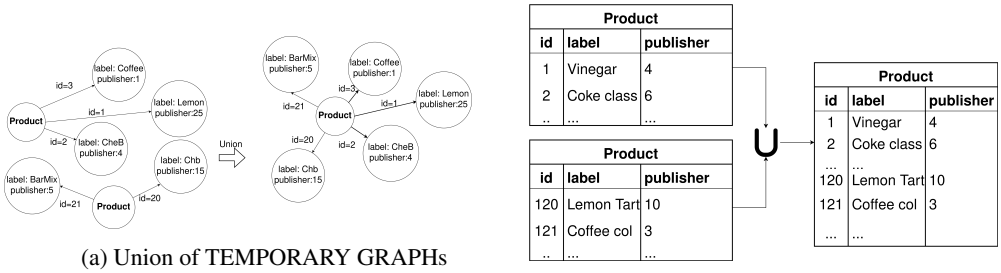
Fig. 3: Parallel Mechanism for GRAPH and TABULAR

We highlighted below how SPARQL and star-shaped queries operations are translated into Virtual Data model operations in case of GRAPH and TABULAR.

3.4 Distributed Query Processing

Distributed Query Processing is where the virtual model is actually joined and executed. Our approach uses Big Data engines (e.g., SPARK) that offer users the ability

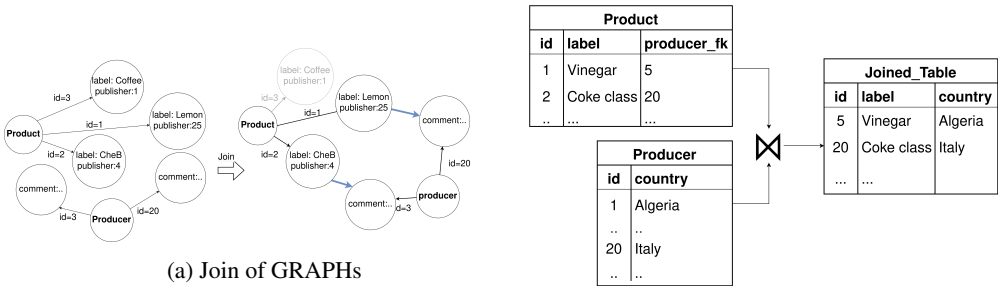
Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216



(a) Union of TEMPORARY GRAPHs

(b) Union of TEMPORARY TABULARs

Fig. 4: Union Operation of TEMPORARY Virtual Model



(a) Join of GRAPHs

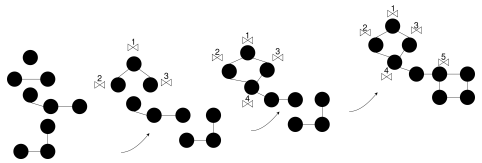
(b) Join of TABULARs

Fig. 5: Join Operation of Virtual Model

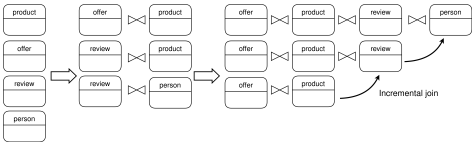
to manipulate the data model of its computation unit (i.e., virtual data model). This allows the implementation of different data models that can be more suitable for various queries. We consider two types of data models, GRAPH and TABULAR, which allow for graph-parallel (see Figure 3a) and data-parallel (see Figure 3b) computation, thus affecting the query performance. Our approach uses several different data models (property graph, document-based, wide-columnar, relational, and tabular) to demonstrate its capability to cover and access various heterogeneous data sources. We should point out that we did not employ any query optimization function to choose the most efficient query execution plan; instead, we focused on the join operation. For instance, if our predictive model predicts based on the input SPARQL query that the optimal virtual model is of type GRAPH, then for each relevant entity, one virtual GRAPH model is generated, following our proposed transformation process (see Subsection 3.3). Once generated, our approach joins those GRAPHs or TABULARs (i.e., a virtual model for each relevant entity) into a FINAL Virtual, GRAPH, or TABULAR (see Figure 5). Below we describe the join process and operations using GRAPH or TABULAR virtual models.

Joining Virtual Data Model: The data join coming from different data sources are represented actually by the connections between star-shaped queries i.e., two SSQs st_a, st_b (subject, predicate, object) are connected if the object of st_a is the subject of st_b. These connections are translated into an array of join pairs (see green SSQ

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216



(a) Multi-Join Algorithm of GRAPHs



(b) Incremental Join Algorithm of TABULARs [12]

Fig. 6: Join Algorithms for GRAPH and TABULAR

in Figure 4a).As for GRAPH, the FINAL Virtual GRAPH (Figure 5a) is created by iterating through the GRAPHs join pairs following a multi-way join algorithm (Figure 6a) which has been proven beneficial in terms of performance based on research literature [29]. The multi-way join algorithm can join two or more relations simultaneously, which is suitable for graph-parallel computation. In practice, new edges are created for each joined pair to link GRAPHs, such as an edge source point to one of the GRAPH vertices and its destination points to the second GRAPH. The FINAL Virtual GRAPH is the result of the newly created edges and the union of the joined pair vertices. Finally, we filter out vertices' identifiers that have no destination. Furthermore, to make the joining of GRAPHs faster, we selected only projection columns' IDs before joining GRAPHs since it is heavy to scan over columns. Similarly, the FINAL Virtual TABULAR i.e., joined TABULARs (Figure 5b) is created by applying join between the respective tables following incrementally joined (Figure 6b), which is revealed to be very efficient [30]. This is done by using a predefined method `'join'` that takes the joined pairs' names and the name of the foreign key column as an argument. Furthermore, we adopted the same strategy proposed in [24], which employs a filter before data transformation, thus reducing the number of the values of the attributes to be transformed and then joined which revealed high efficiency.

- **Star-Shaped/SPARQL Operations to GRAPH/TaABULAR Operations** GRAPH and TABULAR have different structures; therefore, the interaction with GRAPH is possible through Graph Pattern Matching operations (e.g., Cypher-like), while the interaction with TABULAR is possible through SQL-like functions. We highlighted below how SPARQL and star-shaped operations are translated into Virtual Data model operations, GRAPH, and TABULAR.

  – Projection: this operation requires accessing FINAL Virtual GRAPH and TABULAR. For GRAPH, we used the hash map method to get the properties' indexes by iterating over the projected vertices and collecting the linked vertices into one vertex. This helps reduce the operations (e.g., limit) execution time by executing operations

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

on a single vertex instead of multiple vertices. Contrary to the FINAL TABULAR, which is projected using a predefined method `'project'` that takes as an argument the projection variables and returns a projected FINAL TABULAR.

- Filtering: Performing filtering on a given property of Virtual GRAPHS needs accessing data through an index rather than the property name. Therefore, we used a hash map that stores the property name and index. We get the right property index by matching the property name from the filter with the one from the hash map. As for the Virtual TABULAR model, filters are executed over the TABULAR columns. We use a predefined method `'filter'` that takes as an argument the filter statement and returns a filtered virtual TABULAR model.

- Ordering and Limit: to be able to sort or show a limited number of data of the GRAPH, we extracted triples from the FINAL GRAPH. Next, we used a predefined ordering method, e.g.,`'sortBy'` and limited method `'take'`, that takes the vertex property value as input and outputs sorted or limited FINAL GRAPH. As for the TABULAR model, it can be sorted and limited using predefined methods `'orderBy'` and `'limit'` respectively. These methods take the ordering column or number of needed rows in case of Limit as an argument and return an ordered or limited FINAL TABULAR.

3.4.1 Query Execution

Optimizing query execution time is a very crucial step when it comes to loading and joining data. However, time optimization depends not only on the virtual data model, i.e., GRAPH or TABULAR, but also on the execution plan of operations, e.g., applying a filter before joining data. We disabled any query optimization by engine Apache SPARK and Graphx to emphasize the join operation when querying multiple data sources.

Optimization Strategy for GRAPH. To join GRAPHs, we applied a multi-way join algorithm (Figure 6a) which has been proven beneficial in terms of performance based on research literature [29]. The multi-way join algorithm can join two or more relations at the same time, which is suitable for graph-parallel computation. Furthermore, to make the join of GRAPHs faster, we selected only projection columns and their ID before joining GRAPHs since it is heavy to scan over columns (unlike the TABULAR strategy given next).

Optimization Strategies for TABULAR. To join TABULARs, research has proven that incremental data processing approaches [30] for data-parallel achieve better performance since they rely on updating the results of a query when updates are streamed rather than re-computing these queries and may require less memory than batch processing. Therefore, we followed the incremental join; if TABULAR is selected as an optimal virtual data model based on query behavior, the FINAL Virtual TABULAR is created by iterating through the TABULARs that are created from the relevant entities and incrementally joined (see Figure 6b). Furthermore, we adopted the same strategy as described by [24] where we applied a filter before data transformation, thus reducing the number of the values of the attributes to be transformed and then joined, which revealed very efficient.

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

4. Deep Learning Model

This section describes our deep learning model to predict the virtual data model of type GRAPH or TABULAR.

4.1 SPARQL Features Analysis

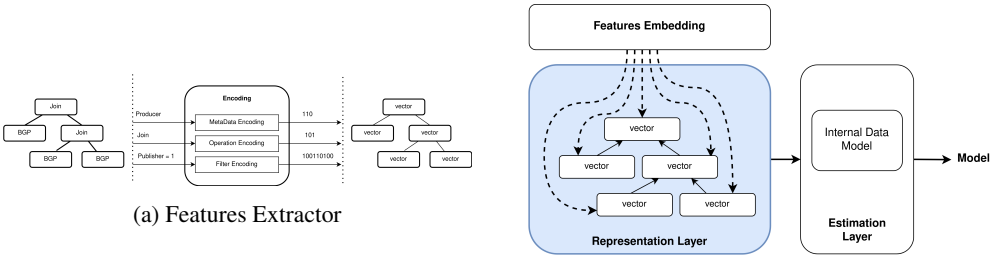
Our model breaks down the SPARQL query plan into nodes (Figure 7a). Each node includes a set of query features that significantly affect the query cost (e.g., filter). The different features are then encoded using different encoding models. Below, we list those features and their encoding:

- **MetaData**: is the set of attributes and entities used in the SPARQL query (e.g., entity names 'producer'). We encode both attributes and entities using a one-hot vector. Then we concatenate each attribute vector with its entities vectors to have a final MetaData vector.
- **Operation:** is the set of physical operations used in the SPARQL query, such as Join, BGP, Projection, OrderBy, and Limit. Each operation is composed of an operator (e.g., ">=") and a list of operands (entities or attributes e.g., [operator='project', attributes='price, delivery-days']). Both the operator and its operands are encoded using a one-hot vector. Finally, each operation vector in the SPARQL query is the concatenation of an operator vector and its operands vectors.
- **Filter:** is the set of query filters. A filter is considered a special operation since it could be either atomic or compound. Each atomic filter is composed of an attribute, an operator, and an operand. The filter operand could be either a float or a string value. Both the attribute and the operator are encoded using a one-hot vector. To encode the operand, we use a normalized float if its value is numeric; otherwise, we use a String representation. The String representation makes use of a Char Embedding model and a CNN (Convolutional Neural Network [31]) to have a fixed-length dense String vector. The three resulting vectors are concatenated to form one single filter vector.
  The compound filter is a combination of multiple atomic filters using either AND or OR operator. For example, 'price $>$ 4000 (atomic) AND price $<$ 20 000 (atomic)', in this case, the filter is considered as a compound. To obtain the vector of the compound filter, we encode each logical operator and atomic filter using one-hot encoding. Next, a tree filter is created where the root is the one-hot vector of a logical operator (e.g., AND), and the nodes are the one-hot vectors of atomic filters (e.g., left node 000111 representing price $>$ 400). Finally, each node (one-hot vector) is transformed into a sequence using the Depth First Search algorithm (DFS). At the end of each sequence, we add an empty node. The sequences are then concatenated following the visited order.

4.2 Proposed Tree-structured Model

Tree-structured models have been proven more powerful than neural networks at predictive tasks using tabular data [32]. Inspired by the work presented in [33], we propose our deep learning model (Figure 7b) that takes as input the encoded features of SPARQL

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216



(a) Features Extractor

(b) Tree-Structured

Fig. 7: Deep Learning: Feature Extraction and Tree-Structured Model

query and outputs the optimal virtual data model, GRAPH or TABULAR that has the lowest cost. Our model consists of an embedding layer to condense the features' vectors and an estimation layer to estimate the optimal virtual data model. In addition, the model includes an intermediate representation layer to capture the correlation between the joined star-shaped queries. In the following, we give a detailed explanation of the model architecture.

4.2.1 SPARQL Embedding Layer

The SPARQL Query Embedding Layer (Figure 8a) embeds a sparse vector into a dense vector. It takes as inputs three types of feature vectors: MetaData, Operation, and Filter. First, the MetaData vector along with the Operation vector are embedded using a fully connected neural network layer[7] with ReLU (Rectified Linear Unit) activator, which is a piecewise linear function that outputs the input directly if it is positive. The structure of the Filter vector is more complicated in the case of compound filters. Therefore, we adapted a Min-Max-Pooling operation to embed the Filter vector.
The Min-Max-Pooling model is a tree-structured model that takes the structure of the Filter tree. For leaf nodes, we use a fully connected neural network. For conjunction nodes, we use the max pooling layer for 'OR' operator and the min pooling layer for 'AND' operator. The max pooling layer is the maximum number of estimated results satisfying the atomic predicates, while the min pooling layer is the minimum number of estimated results satisfying the atomic predicates. Thus representing the SPARQL query filters explicitly.

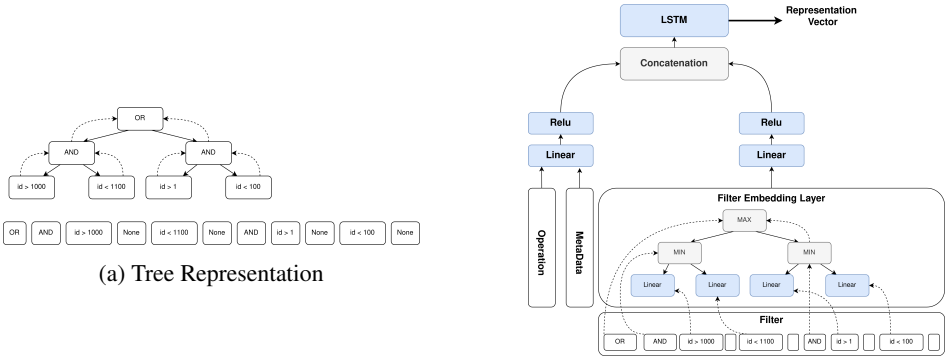
4.2.2 SPARQL Representation Layer

Learning complex structure representations such as tree structure using classic neural networks has many challenges. First, the neural networks can learn much information from the leaf nodes but fails at capturing the correlation among upper nodes. This is known as the vanishing gradient problem. Second, capturing correlations between multiple tree nodes requires storing a lot of intermediate results, which leads the space to grow exponentially. This is known as the gradient explosion problem. To handle those

[7]Fully connected network: a linear regression a0x +..+ anx, 2 linear regressions connected means the output of the first one is the input of the second one.

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

two problems, we designed an intermediate layer (detailed in Figure 8b) that captures the global cost information from leaf nodes to the root by training representations for nodes recursively. We use fully connected networks that have the same structure and share common parameters. Each layer has three inputs: an embedding vector, a representation vector of the right child, and a representation vector of the left child. We used Long Short-Term Memory (LSTM) [34] as a recurrent model. The LSTM model uses the concept of 'memory' to store information of previous nodes, which makes them capable of learning order dependence in the tree structure. This helps prevent the information loss problem. On the other hand, the forget gate of Sigmoid helps LSTM to address the space explosion problem.



(a) Tree Representation

(b) Representation Model

Fig. 8: Deep Learning: Tree and Representation Model

4.2.3 Virtual Model Classification Layer

It is a binary classification model that takes the representation vector of query tree nodes as input and outputs the optimal virtual data model, GRAPH or TABULAR, with the lower cost (i.e., we set GRAPH with value 1 for SPARQL queries that are faster than TABULAR and label TABULAR with value 0 for SPARQL queries that are faster than GRAPH). The classification layer includes two fully connected neural networks with a ReLU activator. The output layer is a Sigmoid function that returns a number from 0.0 to 1.0, representing the probability that the input belongs to. If the output is closer to 1.0 then the predicted virtual data model is of type GRAPH; otherwise, if the output is closer to 0.0, then the predicted virtual data model is of type TABULAR.

	Product	**Offer**	**Review**	**Person**	**Producer**
database type	**Cassandra**	**MongoDB**	**CSV**	**Neo4j**	**MySQL**
# of tuples	50000	50000	50000	50000	50000
data size	~90MB	~4MB	70MB	~3MB	14MB

Table 1: table 1a: Data & Queries Characteristics

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
Product	✓	✓		✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓	✓	✓	✓
Offer	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓		✓	✓
Review	✓	✓	✓	✓		✓	✓	✓		✓			✓	✓	✓			✓	✓	✓
Person			✓	✓		✓			✓		✓	✓						✓	✓	✓
Producer	✓		✓	✓		✓	✓	✓			✓				✓			✓	✓	✓
PROJECT	✓16	✓5	✓29	✓45	✓24	✓45	✓38	✓38	✓24	✓34	✓4	✓6	✓32	✓34	✓4	✓5	✓9	✓45	✓45	✓5
FILTER	✓16		✓12	✓1		✓5	✓1	✓1					✓1	✓1	✓4			✓2	✓3	
ORDERBY	✓1	✓1			✓1		✓1		✓1	✓1			✓1	✓1			✓1			
LIMIT	✓300	✓2		✓20	✓4	✓20	✓20	✓80			✓10		✓13	✓19	✓1000	✓1000				
DISTINCT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 2: Tables and Operations involved in Queries.

5. Implementation and Experimental Setup

OPTIMA - an implementation of our approach, is an OBDA system that calls Graphx and Apache-Spark[8] to implement two virtual data model, GRAPH and TABULAR. The virtual data model is the model defined by the computation unit of these two query engines[9]. Graphx and Apache-Spark already implement wrappers called connectors, of which we used five types to load data that is stored in Neo4j (property graph), MongoDB (document-based), Cassandra (wide-column), MySQL (relational), and CSV (tabular). As for transformation, we used Graphx and Apache-Spark functions[10] e.g., `flaMap(x=>y)`. OPTIMA calls a deep learning model to get the predicted optimal virtual data; it uses NumPy for encoding data and PyTorch for the prediction model. **OPTIMA is available on GitHub at https://github.com/chahrazedbb/OPTIMA**.

We conducted an empirical study to evaluate OPTIMA performance with respect to the following sub-research questions of our problem: RQ1: What is the query performance using OPTIMA? RQ2: Is the time of prediction plus the time of query execution using an optimal virtual model equal to the fixed one? RQ3: What is the query performance when using TABULAR versus GRAPH? RQ4: What is the accuracy of OPTIMA and machine learning? RQ5: What is the query performance of OPTIMA compared to the state-of-the-art, e.g., Squerall [12]? RQ6: What is the impact of involving more data sources in a join query? RQ7: What is the resource consumption (CPU, memory) of OPTIMA while running various queries? RQ8: What is the time taken by each transformation process?

5.1 Benchmark, Queries, and Environment

There is no benchmark dedicated to assessing ontology-based big data access systems. We end up using BSBM* [12] to evaluate the performance of OPTIMA. BSBM* is an adapted version of BSBM benchmark [35] where five tables, Product, Offer, Review, Person, and Producer, are distributed among different data storage. To test OPTIMA, we use the five tables to enable up to 4-chain joins. These tables are loaded in five different data sources Neo4j, MongoDB, Cassandra, MySQL, and CSV. Table 1 shows the described information about data. We generated 5150 queries with 0-4 joins, 0-45 selection, and 0-16 for the filter, limit, and orderBy. The characteristics of these queries

[8] for Apache-Spark, a small part of OPTIMA is based on Squerall's code (https://github.com/EIS-Bonn/Squerall)

[9] RDD is an immutable distributed collection of elements, while DataFrame is an immutable distributed collection of data organized into named columns. RDD is distinct from DataFrame in that the former is considered schema-less.

[10] https://spark.apache.org/docs/latest/graphx-programming-guide.html, https://spark.apache.org/docs/2.2.0/rdd-programming-guide.html

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

Query	OPTIMA	Squerall
SELECT DISTINCT ?productLabel ?producerLabel WHERE { product rdfs:label ?productLabel . ?producer rdfs:label ?producerLabel . ?product rdf:type bsbm:Product . ?product bsbm:producer ?producer .}	{'Bar Mix Lemon','Coke Classic 355 Ml'}	{'Bar Mix Lemon','Coke Classic 355 Ml'}

(a) Query Result Returned by OPTIMA & Squerall

System	Time (ms)
OPTIMA	2400
Squerall	4200

(b) Avg Time

Metrics	**OPTIMA**	**Squerall**
CPU average (%)	0.21	0.20
Max memory (GB)	1.0	0.97

(c) Resource Consumption

Table 3: OPTIMA Performance

are presented in Table 2. We take 4120 queries for training the model and 1030 queries for validation. We run the evaluation on Ubuntu Version 20.04 64-bit with an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, allocating 8GB of RAM.

Training paradigm

In this section, we provide a detailed description of the training paradigm of our deep learning model. The training data typically involves the following steps:

- Data collection and preprocessing: To the best of our knowledge, no large datasets of SPARQL queries exist. Therefore, we generated more than 5000 SPARQL queries that combine all possible elements of a SPARQL query, as described in Table 1. These queries are then preprocessed (see an example of SPARQL query in Appendix A.1) to extract features (see Appendix A.2) and then convert them into a tree-structured representation (see Appendix A.3) suitable for input into our deep-learning model. We run each query on both GRAPH and TABULAR. We set GRAPH with a value of 1 for SPARQL queries that are faster than TABULAR and label TABULAR with a value of 0 for SPARQL queries that are faster than GRAPH.

- Tree construction: The tree structure is constructed based on the query plan, in other words, into query result clause and query pattern and query. For example, the tree's root node represents the query plan, while the child nodes represent the query result clause and query pattern and query, and the leaf nodes of the query result clause would represent clause type such as the "SELECT" operation (see Appendix A.3).

- Supervised learning: To enable the model to learn the relationships between the SPARQL query elements (e.g., plan, operators, etc.) and the execution time of each data model GRAPH or TABULAR. We trained our deep learning model using feed-forward neural network with multiple hidden layers and non-linear activation functions ReLU and Sigmod, including two fully connected neural networks, each with 16 neurons. We trained the model on 80% of queries using the mean squared error as the loss function and the Adam optimization algorithm. The model is trained for 100 epochs, and the validation loss is monitored to prevent overfitting.

- Model evaluation: We evaluated our trained model's accuracy, and we obtained good results after iterations.

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

System	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
OPTIMA	1291	1254	730	10299	10199	1553	7104	8442	10094	4694	2575	233	4673	4487	2397	2881	1698	4607	2804	5648
Squerall	4098	2519	3091	10283	10191	7984	7089	8427	10088	4684	2561	1400	4644	4469	3885	2875	3314	8742	9059	7407
Time Difference	2807	1265	2361	16	8	6431	15	15	6	10	14	1167	29	18	1488	6	1616	4135	6255	1759

Table 4: Time in ms per Query of OPTIMA & Squerall

5.2 Metrics

To evaluate OPTIMA, we use the following metrics:

- OPTIMA Accuracy. We compare the results returned by OPTIMA against the results returned by Squerall.
- Classification. We use two metrics to evaluate the OPTIMA classification model: Cross-entropy loss and Accuracy function. Assuming the real result is denoted as $r = r_i$, the predicted result is denoted as $p = p_i$, and the correctly predicted results as $tp = tp_i$, where $1 <= i <= N$, we compute these metrics as follows: $CE(r,p) = \sum ri * log(pi)$, $Acc(tp,p) = \sum tp_i / \sum p_i$
- Memory and CPU consumption as described in [36]. Specifically, we measure how much the memory and CPU are active during the computation.
- Execution Time. We measure the time OPTIMA takes from query submission to the delivery of the answer. The time is measured using the absolute wall-clock system time reported by the Scala time() function.

5.3 Method

We consider two studies:

- In the first study, we compare OPTIMA's results with SPARK-based Squerall's results. Our comprehensive literature review did not reveal any single work except Squerall that is available and that supports most data sources. Squerall uses two big data engines, Presto and SPARK: Presto-based, where the virtual model of presto engine (which cannot be controlled by users) is used for query processing, and SPARK-based, where DataFrames are created as a virtual data model. To make the results comparable, we choose SPARK-based Squerall and extend it to support Neo4j. We assess the accuracy of OPTIMA in terms of (1) results (accuracy), (2) time, and (3) CPU and memory usage compared to SPARK-based Squerall. We should note that comparing the overall execution time of OPTIMA against an original system, e.g., relational for a given query, is impossible because we are querying various heterogeneous formats and models.
- In the second study, we inspect OPTIMA's main components: machine learning, data wrappers, and query execution. We observed the behavior of query execution for GRAPH and TABULAR in terms of time. For the data wrapper, we investigate the time taken for the transformation process from data sources to GRAPH or TABULAR. As for the machine learning component, we compare our model with the LSTM model in terms of accuracy and time. The LSTM model takes as input the encoded features vectors without any correlation and outputs the data model.

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

5.4 Experiment 1: OPTIMA vs SPARK-based Squerall

In this experiment, we load BSBM* as described above to obtain the results from OPTIMA and SPARK-based Squerall. Then, we run 5150 SPARQL queries and compare the results.

- **Validation of Results and Overall Execution Time:** this comparison allows us to confirm the correctness of the results returned by OPTIMA. Table 3a shows the results of OPTIMA and SPARK-based Squerall of a complex SPARQL query Q21. The results are the same for both systems, which confirms that OPTIMA is able to support and join large data coming from different datasets.

  Table 4 illustrates the execution time returned by both systems. As can be observed, OPTIMA excels Squerall for queries that involve multiple joins. The time difference ranges from 0 to 80000 milliseconds (ms). This difference is due to the predicted virtual data model e.g., Q19, Q20, in which deep learning predicted that the Virtual model of type GRAPH is optimal. We also observe a small difference in the execution time (ranging from 0 to 30 ms) in favor of Squerall compared to OPTIMA for queries that involve multiple projections e.g., Q7, Q10. This is explained by the fact that the optimal virtual model is identical to Squerall's, and both Squerall and OPTIMA used the same APIs to call data (wrapper); however, the data model prediction time added to OPTIMA makes it slightly slower than Squerall. Furthermore, the average execution time of Squerall is greater than 4000 ms compared to the average execution time of OPTIMA 2400 ms as shown in table 3b. These results illustrate the benefits of OPTIMA over existing systems; thus, RQ1 and RQ5 are answered.

- **Data Model Execution Time.** As shown in Table 5, the analysis of experimental results indicates that GRAPH is faster than TABULAR in most cases, except for queries like Q8 and Q10. It has comparable to slightly lower performance in Q16. This confirms that the optimal model is very important in reducing the execution time of queries. The total execution time ranges from 50 to 90000 ms, with 90% of all cases being about or below 3000 ms. OPTIMA virtual data model of type GRAPH is faster in queries that involve joins (ranging from 50 to 40000 ms), while the TABULAR model outperforms the GRAPH model in queries involving more projections (ranging from 200 to 90000 ms).

  This is explained by the fact that the GRAPH is designed to store connections between data. Therefore, queries do not scan the entire graph to find the nodes that meet the search criteria. It looks only at nodes that are directly connected to other nodes, while SQL-like methods used by the TABULAR model require expensive join operations because they traverse all data to find the data that meets the search criteria. On the other hand, the TABULAR model is faster when handling projections because the data structure is already known, and data can be easily accessed by column names. Conversely, the GRAPH model does not have a predefined structure for the data, and each node attribute has to be examined individually during the projection query.

  The number of joins has a decisive impact on query performance; it should be taken into consideration with other factors, e.g., size of involved data, presence of filters, and selected variables. For example, Q2 joins only two data sources, Product and Review

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20
Prediction Time	3	3	4	6	4	5	5	6	2	4	5	1	5	5	4	3	2	4	4	4
GRAPH	1143	1161	1239	1243	306	3181	7168	12237	4977	16681	1211	3567	482	1285	766	2883	6639	1366	3370	1723
TABULAR	4098	2519	3091	10283	10191	7984	7089	8427	10088	4684	2561	1400	4644	4469	3885	2875	3314	8742	9059	7407

Table 5: Time in ms per Query of Prediction, GRAPH & TABULAR

(1254 ms) but has comparable performance with Q1 (1291 ms), which joins four entities (Product, Offer, Review, and Producer). This may be due to filtering in Q1 (16 filters), significantly reducing intermediate results to join. Q3 involves four data sources, yet it is among the fastest queries. This is because it involves the small entities Person and Producer, which is another reason to reduce intermediate results to join. With five data sources to join, Q4 is among the most expensive queries (10299 ms). This can be attributed to the fact that the filter on Product is selective (?language = "en"), which results in large intermediate results to join, in contrast to Q6 ($?price < 8000$). Although the four-source join Q7 and Q8 involve the small entity Producer, they are the most expensive queries that execute over the GRAPH model; this can be attributed to a large number of projections (38 attributes). Thus, we answer RQ3 and RQ6 and suggest that operations can affect query execution time.

- Resource Consumption: finally, we record the Resource Consumption (i.e., Memory and CPU) taken by OPTIMA and SPARK-based Squerall. The results reported in Table 3c show that the CPU is not fully used by OPTIMA and SPARK-based Squerall (around 0.21% was used). This means that the complexity of queries does not impact CPU consumption. As for the total memory reserved, OPTIMA consumed around 1GB over 8GB per node, while SPARK-based Squerall used at most 1GB. Having the same CPU and memory could be explained by the fact that both are using the same query engine - SPARK, and the distribution of CPU between the nodes for loading and transformation. This answers RQ7.

5.5 Experiment 2: Performance of OPTIMA's Predictive Model

In this study, we evaluate the main components of OPTIMA.

- **Deep Learning Accuracy.** We evaluated our model with LSTM and Regression models to assess our encoding techniques and prediction model. We used 5150 queries; 80% for training and 20% for validation. We trained all models on the same dataset and computed the accuracy and Cross-entropy loss function. Results in Table 6a show that our tree-structure-based method outperforms the LSTM and Regression models with an average accuracy of 0.831 for our model against 0.708 and 0.717 for LSTM and Regression, respectively. The cross-entropy loss is equal to 0.00018 for our model compared to 1.92027 and 6.51098 for LSTM and Regression, respectively. This is explained by the fact that both models, LSTM and Regression, rely on the independent assumption among different operations and attributes, while our model achieves the best performance as it captures more correlations. Thus answering RQ4.

- **Deep learning reduces the overall execution time.**

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

Cost	Loss	Accuracy
Regression model	1.92027	0.708
LSTM	6.51098	0.717
Our Model	0.00018	0.831

(a) Loss & Accuracy of Deep Learning Models

Condition	**Avg. time (ms)**
Machine Learning	12
Only GRAPH	1320
Only TABULAR	2862

(b) Time of Deep Learning, GRAPH & TABULAR

Table 6: Deep Learning Performance

To check if deep learning is reducing the overall execution time of OPTIMA by selecting the optimal virtual data model. We illustrate first the time taken by OPTIMA's components: machine learning algorithm, query execution over GRAPH model, and query execution over TABULAR against SPARK-based Squerall. We run OPTIMA and Squerall over 1030 queries. Results are shown in table 6b. The average execution time of the machine learning component is a very short 12 ms, while the average time for GRAPH is 1320 ms and TABULAR is 2862 ms. Results show that for most queries, GRAPH is faster than TABULAR, even with prediction time. In summary, only 14% of the queries were initially faster for OPTIMA (using GRAPH as a virtual model) compared to Squerall and become in the later favor. This is explained by the fact that for those queries, there is a slight difference in execution time using GRAPH compared to Squerall. This answers RQ2.

Model	**Neo4j**	**JDBC**	**CSV**	**Cassandra**	**MongoDB**	**Loading**
GRAPH	138	954	196	7695	188	4.327
TABULAR	3275	199	255	5319	330	7.141

Table 7: Time (ms) of Data transformation to GRAPH & TABULAR

5.5.1 Data wrapper Time

To answers RQ8, we evaluate, in this study, the time needed to load the data from data sources to the virtual data model of type GRAPH or TABULAR (see Table 7). Since the transformation process is different, we expect different behavior from the wrappers. In the table, we illustrate the time needed by each wrapper with the following observations:

- Neo4j connector loads 50000 nodes from Neo4j within 138 ms into GRAPH, compared to 3275 ms in TABULAR. This is explained by the fact that the graph property used by Neo4j has the same exact structure as the GRAPH model.
- CSV connector loads 50000 rows within 196 ms from CSV files into GRAPH, compared to 255 ms in TABULAR, even though CSV files save data into tables. This can be explained by the fact that GRAPH virtual model is a schema-less model that loads data directly without the need to preserve data structure, while TABULAR takes time to build the data schema.
- JDBC connector loads 50000 rows from MySQL database within 954 ms into GRAPH, compared to 199 ms in TABULAR. This can be explained by the fact that MySQL uses a relational model, which has the same data structure as the Virtual TABULAR model.

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

- MongoDB connector loads 50000 rows from MongoDB within 188 ms into GRAPH, compared to 330 ms in TABULAR. This can be explained by the fact that MongoDB is document-based i.e., it is schema-less, the same as the GRAPH Virtual model, unlike the TABULAR model, which needs to build a data schema.

- Cassandra connector loads 50000 rows within 7695 ms into GRAPH, compared to 5319 ms in TABULAR. This can be explained by the fact that Cassandra uses a columnar data model, which is more close to the TABULAR model even though it is a NoSQL database.

6. Related Work

Our literature review reveals two categories addressing data virtualization. These two categories are namely "ontology-based data access" and "non-ontology-based data access" [12]. Non-ontology-based data access approaches mostly use SQL-like as query language and implement a virtual relational model [37,38], defining views of relevant data from sources having a relational model. Those views are generated based on mapping assertions that associate the general relational schema with the data source schemata. The shortcomings of these approaches are that the schema modifications and extensions are very rigid due to mappings and may depend on complex constraints. Furthermore, these approaches use Self-Contained Query [24] where users cannot control the structure of the virtual data model. OBDA [39] approaches use SPARQL as a unified access language and detect relevant data from sources to be joined through ontologies and standardized mappings. This provides flexibility in modifying and extending the ontology and mappings with semantic differences found across the data schemata.

Exiting Systems implemented OBDA over relational databases, e.g., Ontop [40], Stardog (http://www.stardog.com), which are using virtual knowledge graphs. These solutions are not designed to query large-scale data sources, e.g., NoSQL stores or HDFS. Our study's scope focuses on works that query large-scale data sources using OBDA. Optique [10] is an OBDA platform that accesses both static and streaming data. It implements a relational model (implicitly a TABULAR) as a virtual model while querying data sources such as SQL databases and other sources e.g., CSV, and XML. There was no clear description of how Optique accesses NoSQL stores and distributed file systems (e.g., HDFS). Ontario [11] focuses on query rewriting, planning, and federation, with a strong stress on RDF data as input. Query plans are built and optimized based on a set of heuristics. The virtual model used by Ontario is the GRAPH model (explicitly an RDF). Squerall [12], recent and close work to OPTIMA leverages Big Data engines SPARK and Presto to query on-the-fly heterogeneous large data sources. The virtual data model imposed by Presto is TABULAR and does not offer users to control it, while SPARK can offer control over the virtual data model, Squerall uses DataFrame as a virtual model which is TABULAR. However, the decision behind the virtual data model implemented by all these systems is rather based on use and flexibility and not on solid evidence to improve query processing. There is no work that (1) implements the different optimal virtual models, and (2) selects the optimal one based on query behavior. For machine learning, some works [14,15,16] addressed the cost estimation of SPARQL queries to op-

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

timize query execution plan e.g., performance prediction, however, all these approaches are designed for a single query on one single data source.

7. Conclusion

We presented a new approach that reduces the time execution of querying large heterogeneous data by predicting the optimal virtual data model based on query behavior. OPTIMA - a realization of our approach, implements two virtual models, GRAPH and TABULAR within the query engine SPARK (Graphx and Apache-Spark). The effective deep learning model built on top of OPTIMA's architecture estimates the cost of the query against both virtual models to select the optimal one for the given query. It extracts significant features such as the query plan and query operation and returns the optimal virtual data model. Once selected, OPTIMA gets a unified view of the data from multiple data sources on-the-fly by decomposing the input SPARQL query into star-shaped queries. Next, it uses ontology-based mappings to detect relevant entities from original sources. Those relevant entities are then loaded by the wrappers into the predicted virtual model, GRAPH or TABULAR to be joined. In the case of GRAPH, a set of vertex and edges are joined while for TABULAR, a set of tables is combined. Finally, the results are returned by applying the operation on the FINAL joined GRAPH or TABULAR. Extensive experiments showed a reduction in query execution time of over 40% for the TABULAR model and over 30% for the GRAPH model.

A. Appendix: Training Data

In this appendix, we present an example of the data collection and preprocessing of the training model.

```
SELECT DISTINCT  ?vendor ?country ?producerPublisher
WHERE{
        ?producer edm:country ?country.
        ?producer bsbm:publisher ?roducerPublisher.
        ?offer bsbm:producer ?producer.
        ?offer bsbm:vendor ?vendor.
        ?offer rdf:type schema:Offer.
        FILTER (?country = "DE" ).
        FILTER (?producerPublisher = "1" ).
        FILTER (?vendor >= 50 ).
}
```

Listing A.1: SPARQL query

```
A.[DISTINCT,B,H,I,J,K]
B.[PROJECT,?vendor,?country, producerPublisher]
C.[BGP,D,E]
D.[TRIPLE,?producer,?country]
E.[TRIPLE,?producer,?producerPublisher]
```

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216


```
F.[BGP,G]
G.[TRIPLE,?offer,?vendor]
H.[JOIN,C,F]
I.[FILTER,    =    ,?country,"DE"]
J.[FILTER,    =    ,?producerPublisher,"1"]
K.[FILTER,    >=    ,?vendor,50]
```

Listing A.2: Feature extraction from SPARQL queryA.1

```
{
 "Plan":
        {
            "DISTINCT":""
            ,"PROJECT":
            [
                "?vendor"
                ,"?country"
                ,"?producerPublisher"
            ]
            ,"JOIN":
            {
                "TYPE" : "left"
                ,"BGP" :
                {
                    "TABLE" : ["producer"]
                    ,"TRIPLE" : ["?producer","?country"]
                    ,"TRIPLE" : ["?producer","?producerPublisher"]
                }
                ,"BGP" :
                {
                    "TABLE" : ["offer"]
                    ,"TRIPLE" : ["?offer","?vendor"]
                }
            }
            ,"FILTER":
            {
                "op_type": "Compare"
                ,"operator": "="
                ,"left_value": "?country"
                ,"right_value": "DE"
            }
            ,"FILTER":
            {
                "op_type": "Compare"
                ,"operator": "="
                ,"left_value": "?producerPublisher"
                ,"right_value": "1"
            }
            ,"FILTER":
            {
```

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216


```
                "op_type": "Compare"
                ,"operator": ">="
                ,"left_value": "?vendor"
                ,"right_value": "50"
            }
        }
    }
```

Listing A.3: Tree Represenation of SPARQl QueryA.1

Acknowledgments

The authors acknowledge the financial support of Fraunhofer Cluster of Excellence (CCIT) and Dr. Mohamed Najib Mami for the valuable comments that helped to implement our work.

REFERENCES

1. D. Age, "2025: The digitization of the world–from edge to core," *Farmingham, MA.*, 2020.
2. C. Snijders, U. Matzat, and U.-D. Reips, "" big data": big gaps of knowledge in the field of internet science," *International journal of internet science*, Vol. 7, no. 1, 2012, pp. 1–5.
3. A. Gandomi and M. Haider, "Beyond the hype: Big data concepts, methods, and analytics," *International journal of information management*, Vol. 35, no. 2, 2015, pp. 137–144.
4. A. Cuzzocrea, L. Bellatreche, and I. Song, "Data warehousing and OLAP over big data: current challenges and future research directions," in *Proceedings of the sixteenth international workshop on Data warehousing and OLAP, DOLAP 2013, San Francisco, CA, USA, October 28, 2013*. ACM, 2013, pp. 67–70.
5. M. N. Mami, "Strategies for a semantified uniform access to large and heterogeneous data sources," Ph.D. dissertation, University of Bonn, Germany, 2021.
6. M. Rouse, "What is data virtualization," 2011.
7. N. Miloslavskaya and A. Tolstoy, "Big data, fast data and data lake concepts," *Procedia Computer Science*, Vol. 88, 2016, pp. 300–305.
8. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, "Linking data to ontologies," *Journal on Data Semantics X*. Springer, 2008.
9. H. Dehainsala, G. Pierra, and L. Bellatreche, "Ontodb: An ontology-based database for data intensive applications," in *Advances in Databases: Concepts, Systems and Applications, 12th International Conference on Database Systems for Advanced Applications, DASFAA Thailand*. Springer, 2007, pp. 497–508.
10. M. Giese, A. Soylu, G. Vega-Gorgojo, A. Waaler, P. Haase, E. Jiménez-Ruiz, D. Lanti, M. Rezk, G. Xiao, Ö. Özçep *et al.*, "Optique: Zooming in on big data," *Computer*, Vol. 48, no. 3, 2015.

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

11. K. M. Endris, P. D. Rohde, M.-E. Vidal, and S. Auer, "Ontario: Federated query processing against a semantic data lake," in *International Conference on Database and Expert Systems Applications*. Springer, 2019, pp. 379–395.
12. M. N. Mami, D. Graux, S. Scerri, H. Jabeen, S. Auer, and J. Lehman, "Squerall: Virtual ontology-based access to heterogeneous and large data sources," *Proceedings of 18th International Semantic Web Conference*, 2019.
13. S. T. Al-Amin, C. Ordonez, and L. Bellatreche, "Big data analytics: Exploring graphs with optimized SQL queries," in *Database and Expert Systems Applications - DEXA 2018 International Workshops, BDMICS, BIOKDD, and TIR, Regensburg, Germany, September 3-6, 2018, Proceedings*. Springer, 2018, pp. 88–100.
14. W. E. Zhang, Q. Z. Sheng, Y. Qin, K. Taylor, and L. Yao, "Learning-based SPARQL query performance modeling and prediction," *World Wide Web*, Vol. 21, no. 4, 2018, pp. 1015–1035.
15. R. Hasan and F. Gandon, "A machine learning approach to sparql query performance prediction," in *International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, 2014, pp. 266–273.
16. R. Singh, "Inductive learning-based sparql query optimization," *Data Science and Intelligent Applications*, 2021, pp. 121–135.
17. I. Zouaghi, A. Mesmoudi, J. Galicia, L. Bellatreche, and T. Aguili, "Query optimization for large scale clustered RDF data," in *Proceedings of the 22nd International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data co-located with EDBT/ICDT 2020 Joint Conference, DOLAP@EDBT/ICDT 2020,Denmark*, 2020, pp. 56–65.
18. E. Gallinucci, M. Golfarelli, and S. Rizzi, "Schema profiling of document-oriented databases," *Information Systems*, Vol. 75, 2018.
19. A. Senk, M. Valenta, and W. Benn, "Distributed evaluation of xpath axes queries over large XML documents stored in mapreduce clusters," in *25th International Workshop on Database and Expert Systems Applications, Germany, 2014*, 2014, pp. 253–257.
20. J. F. Sequeda, M. Arenas, and D. P. Miranker, "On directly mapping relational databases to rdf and owl," in *Proceedings of the 21st international conference on World Wide Web*, 2012, pp. 649–658.
21. M. A. Rodriguez and P. Neubauer, "The graph traversal pattern," in S. Sakr and E. Pardede, (eds.), *Graph Data Management: Techniques and Applications*. IGI Global, 2011, pp. 29–46.
22. M. Y. Santos and C. Costa, "Data warehousing in big data: From multidimensional to tabular data models," in *Proceedings of the Ninth International C* Conference on Computer Science & Software Engineering, Portugal, 2016*. ACM, 2016, pp. 51–60.
23. M. Vidal, E. Ruckhaus, T. Lampo, A. Martínez, J. Sierra, and A. Polleres, "Efficiently joining group patterns in SPARQL queries," 2010, pp. 228–242.
24. M. N. Mami, D. Graux, S. Scerri, H. Jabeen, S. Auer, and J. Lehmann, "Uniform access to multiform data lakes using semantic technologies," in *Proceedings of the 21st International Conference iiWAS2019*. ACM, 2019, pp. 313–322.
25. J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "{GraphX}: Graph processing in a distributed dataflow framework," in *11th USENIX symposium on operating systems design and implementation (OSDI 14)*, 2014, pp. 599–613.

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

26. S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *International Journal of Data Science and Analytics*, Vol. 1, no. 3, 2016, pp. 145–164.
27. D. Crankshaw, A. Dave, R. S. Xin, J. E. Gonzalez, M. J. Franklin, and I. Stoica, "The graphx graph processing system," *UC Berkeley AMPLab*.
28. M.-E. Vidal, E. Ruckhaus, T. Lampo, A. Martínez, J. Sierra, and A. Polleres, "Efficiently joining group patterns in sparql queries," in *Extended Semantic Web Conference*. Springer, 2010, pp. 228–242.
29. M. Henderson and R. Lawrence, "Are multi-way joins actually useful?." in *ICEIS (1)*, 2013, pp. 13–22.
30. I. Elghandour, A. Kara, D. Olteanu, and S. Vansummeren, "Incremental techniques for large-scale dynamic query processing," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018, pp. 2297–2298.
31. K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *European conference on computer vision*. Springer, 2016, pp. 630–645.
32. Y. Yang, I. G. Morillo, and T. M. Hospedales, "Deep neural decision trees," *CoRR*, Vol. abs/1806.06988, 2018.
33. J. Sun and G. Li, "An end-to-end learning-based cost estimator," *Proc. VLDB Endow.*, Vol. 13, no. 3, 2019, pp. 307–319.
34. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, Vol. 9, no. 8, 1997, pp. 1735–1780.
35. C. Bizer and A. Schultz, "The berlin SPARQL benchmark," *International Journal on Semantic Web and Information Systems (IJSWIS)*, Vol. 5, no. 2, 2009, pp. 1–24.
36. D. Graux, L. Jachiet, P. Geneves, and N. Layaïda, "A multi-criteria experimental ranking of distributed sparql evaluators," in *2018 IEEE International Conference on Big Data*. IEEE, 2018, pp. 693–702.
37. R. F. van der Lans, "Architecting the multi-purpose data lake with data virtualization," *Denodo whitepapers*, 2018.
38. D. Chatziantoniou and V. Kantere, "Just-in-time modeling with datamingler," in *Proceedings of the ER Demos and Posters 2021 co-located with 40th International Conference on Conceptual Modeling (ER 2021), Canada*, Vol. 2958. CEUR-WS.org, 2021, pp. 43–48.
39. D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, and R. Rosati, "Ontologies and databases: The dl-lite approach," in *Reasoning Web International Summer School*. Springer, 2009, pp. 255–356.
40. D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, and G. Xiao, "Ontop: Answering sparql queries over relational databases," *Semantic Web*, Vol. 8, no. 3, 2017, pp. 471–487.

Data Intelligence Just Accepted MS.
https://doi.org/10.1162/dint_a_00216

Chahrazed BACHIR-BELMEHDI received the BSc degree in Information System, in 2017 and the MSc degree in Information Systems engineering, in 2019 from the Djillali Liabes University, Algeria. She is working toward the doctoral degree in LabRI laboratory at ESI-SBA, Algeria. Her research interests include Big Data Analysis, Machine Learning and Software Engineering.

Abderrahmane Khiat holds a PhD in Knowledge Engineering from the University of Oran1, Algeria (2017) and currently works as a Senior Researcher at Fraunhofer IAIS, Germany. He ranked fifth among the top young inventors in the Middle East and Africa in the "Stars of Science-2021". He obtained the price of The Best Paper Awards for Young Scientists Researchers in 2014. His research includes Knowledge Graphs, Big Data Integration and Data Mining.

Nabil Keskes has completed Prof (2020), M.Sc.(2006), Ph.D.(2012). Currently he is a full professor in high School of Computer Science in Algeria. His research interests are geared towards web service selection and pragmatic web.