# Compact Representations for Efficient Storage of Semantic Sensor Data

Farah Karim[1][2], Maria-Esther Vidal[1] and Sören Auer[1]

[1]*Leibniz University of Hannover, Welfengarten 1B, 30167 Hannover, Germany*
[2]*Mirpur University of Science and Technology (MUST), Mirpur-10250 (AJK), Pakistan*
{*karim, vidal, auer*}@*l3s.de*

Keywords: Sensor Data, Data Factorization, and Query Execution.

Abstract: Nowadays, there is a rapid increase in the number of sensor data generated by a wide variety of sensors and devices. Data semantics facilitate information exchange, adaptability, and interoperability among several sensors and devices. Sensor data and their meaning can be described using ontologies, e.g., the Semantic Sensor Network (SSN) Ontology. Notwithstanding, semantically enriched, the size of semantic sensor data is substantially larger than raw sensor data. Moreover, some measurement values can be observed by sensors several times, and a huge number of repeated facts about sensor data can be produced. We propose a *compact* or *factorized* representation of semantic sensor data, where repeated measurement values are described only once. Furthermore, these compact representations are able to enhance the storage and processing of semantic sensor data. To scale up to large datasets, factorization based, tabular representations are exploited to store and manage factorized semantic sensor data using Big Data technologies. We empirically study the effectiveness of a semantic sensor's proposed compact representations and their impact on query processing. Additionally, we evaluate the effects of storing the proposed representations on diverse RDF implementations. Results suggest that the proposed compact representations empower the storage and query processing of sensor data over diverse RDF implementations, and up to two orders of magnitude can reduce query execution time.

## 1 INTRODUCTION

Internet of Things (IoT), cyber-physical systems, and sensor data applications are of paramount importance in our increasingly data-centric society and receive growing attention from the research community. RDF representations of IoT data are being generated (Gaur et al., 2015; Jabbar et al., 2017) to add semantics to the data and turn the data into meaningful actions. The Semantic Sensor Network (SSN) Ontology (Compton et al., 2012) is a W3C standard to describe the sensor data, refer as semantic sensor data. The SSN Ontology consists of several classes and corresponding properties to describe the meaning of sensor data in terms of sensor capabilities, observations, and measured values in an RDF graph. However, RDF representations generate an enormous amount of data; thus, efficient representations of sensor data are required. Furthermore, several sensor observations with the same measurement values generate RDF data redundancy, e.g., 13°F temperature observed by several sensors over the different timestamps. These data redundancies negatively impact the size of the semantic sensor data, hence the storage and processing of this data. Therefore, efficient representations of semantic sensor data are required to store and process large amounts of sensor data using different RDF implementations. Rule-based (Joshi et al., 2013; Meier, 2008; Pichler et al., 2010) and binary (Álvarez-García et al., 2011; Bok et al., 2019; Fernández et al., 2013; Pan et al., 2014) compression techniques for RDF data effectively reduce the size of the data. Distributed and parallel processing frameworks for Big Data are exploited in several approaches (Du et al., 2012; Khadilkar et al., 2012; Mami et al., 2016; Nie et al., 2012; Papailiou et al., 2013; Punnoose et al., 2012; Schätzle et al., 2012; Schätzle et al., 2013). Moreover, column-oriented stores (Idreos et al., 2012; MacNicol and French, 2004; Stonebraker et al., 2005; Zukowski et al., 2006) apply column-wise compression techniques, and improve query performance by projecting the required columns. In the context of query processing, efficient SQL query processing techniques based on the factorization of the data are proposed in (Bakibayev et al., 2013). Despite these storage and processing

techniques, the tremendously growing data requires efficient representations to facilitate the storage and processing.

**Our Research Goal:** We tackle the problem of efficiently representing semantic sensor data described using the SSN ontology. Our research goal is to generate compact representations where redundancies are removed. The proposed compact representations enhance the performance of the query engines by scaling up to large sensor data. We aim at determining the impact of the compact representations on semantic sensor data, and the effect of these representations on query processing.

**Approach:** In this work, we propose the *Compacting Semantic Sensor Data (CSSD)* approach for efficient storage and processing of semantic sensor data. The *CSSD* approach is based on factorizing the data and storing only a *compact* or *factorized* representation of semantic sensor data, where repeated values are represented only once. In addition, universal (Ullman, 1984) and Class Template (CT) based tabular representations leveraging the columnar-oriented *Parquet* storage format are utilized to scale up to even larger RDF datasets.

The effectiveness of the proposed factorization techniques are empirically studied, as well as the impact of factorizing semantic sensor data on query processing using *LinkedSensorData* benchmark (Patni et al., 2010). The observed results demonstrate that the proposed factorization techniques are able to effectively reduce the size of semantic sensor data while all the encoded information is preserved, and improves query performance. This article extends our previous work (Karim et al., 2017), where we introduce the factorization techniques for semantic sensor data to scale-up to large datasets. Here, we present techniques for efficient storage of semantic sensor data and conduct an extended analysis and evaluations of the *CSSD* approach. In essence, we make the following contributions to the problem of storing semantic sensor data:

- The *CSSD* approach using factorization techniques;
- Tabular representations of semantic sensor data to scale-up to large datasets;
- SPARQL query rewriting techniques against factorized sensor data; and
- An empirical evaluation of the proposed compact representations demonstrating the effectiveness and efficiency of the *CSSD* approach.

The article is structured as follows: We motivate the research problem in section 2, and review existing work in section 3. A formal description of our approach is discussed in section 4, and tabular representations in section 5. We present the experimental study in section 6 with an outlook on future work in section 7.

## 2 Motivating Example

The *MesoWest LinkedObservation*[1] datasets encompass sensor data containing weather observations during hurricane and blizzard seasons in the United States. Observations incorporate measurements of several weather phenomena, e.g., wind direction, snowfall, wind speed, rainfall, humidity, and temperature. These weather observations from sensors are semantically described using the Semantic Sensor Network (SSN) ontology. These *LinkedObservations* enclose almost two billion RDF triples semantically describing sensor data collected during major active storms in the United States since 2002. The RDF sensor data describing the weather observations during the storm season in the year 2004 contains 108,644,568 RDF triples representing 11,648,607 observations about different weather phenomena, i.e., precipitation, rainfall, wind direction, temperature, and relative humidity. Figure 1a illustrates the RDF graph of sensor data describing pressure, wind direction, rainfall, temperature, and visibility observations, as well as observation timestamps from the MesoWest dataset during the year 2004. The RDF graph depicts 46,341 RDF triples semantically describing 5,149 sensor observations. The RDF triples associated with the same measurement value are represented by the same color nodes and edges in the graph. The RDF triples affiliated with timestamps are also represented with the same color nodes and edges. The RDF graph statistics, shown in Figure 1b, indicate the existence of remarkably redundant interconnectivity among the RDF nodes. The RDF graph and the statistics present that the RDF triples are replicated with the redundant measurement values. Also, each sensor observation is related to seven neighbors in average, i.e., observations are semantically described using seven RDF triples in average.

Figure 1c depicts the number of RDF triples per distinct measurement value within the RDF dataset. Rainfall measurement value, `6.55 cm`, is highly repeated and is related to 15,552 RDF triples, and $113°$ wind direction is the second highly repeated value and is affiliated with 13,941 RDF triples. Likewise, the number of RDF triples related with unique values can be noticed for other climate phenomena, e.g.,

---

[1] `http://wiki.knoesis.org/index.php/LinkedSensorData`

| S# | Parameter | Value |
|----|-----------|-------|
| 1 | Connected Components | 1.0 |
| 2 | Network Centralization | 0.3 |
| **3** | **Avg. # of Neighbors** | **6.4** |
| 4 | Network Density | 0.0 |
| 5 | Multi-edge Node Pairs | 5,149.0 |
| 6 | Network Heterogeneity | 11.1 |

(a) Original **RDF Graph**  (b) **Statistics** of Original RDF Graph

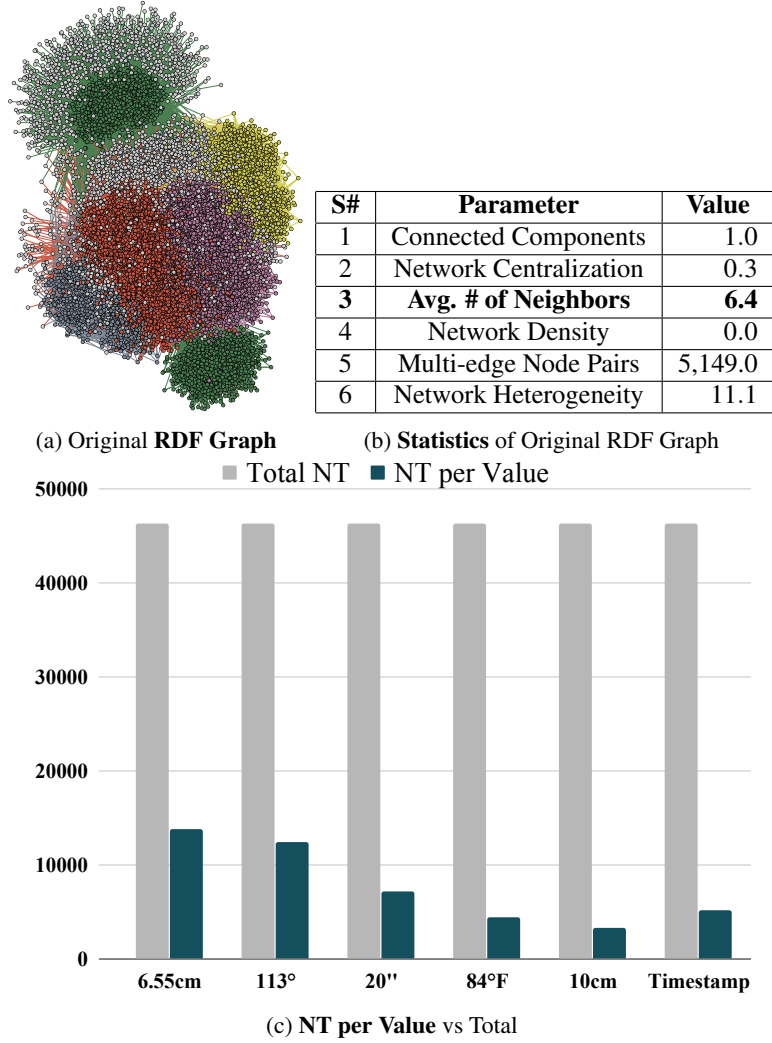(c) **NT per Value** vs Total

Figure 1: **Motivating Example.** (a) An RDF Graph, with the same color of nodes and edges, represents the RDF triples related to same values; (b) Statistics of the RDF graph; (c) Number of RDF triples(NT), associated with same value, to total RDF triples. The RDF graph and statistics are generated by `Cytoscape tool`. (`http://www.cytoscape.org/`).

temperature, pressure, and visibility, and corroborate the *natural intuition* that the number of *observations* is much higher than the the number of *distinct measurement values*. We exploit this natural intuition of semantic sensor data, and present a compact representation. RDF triples of repeated measurements values are *factorized* in these compact representations, and are added to the dataset only once without losing any information initially encoded in the sensor data. Unlike other RDF data compression techniques, the semantics of observations are utilized to factorize the semantic sensor data. The factorized representations provide efficient storage over diverse RDF implementations, and queries can be directly executed over factorized RDF datasets. To scale up to large datasets, tabular representations, based on the factor-

ization, can be used to exploit Big Data technologies for storage and management of large amount of semantic sensor data.

## 3 Related Work

Semantic Web and Big Data communities have been working for better storage and processing of large datasets. RDF compression techniques (Álvarez-García et al., 2011; Bok et al., 2019; Fernández et al., 2013; Meier, 2008; Pan et al., 2014; Pichler et al., 2010) are devised, as well as, Big Data tools are exploited in (Du et al., 2012; Khadilkar et al., 2012; Mami et al., 2016; Nie et al., 2012; Papailiou et al., 2013; Punnoose et al., 2012; Schätzle et al.,

2013) to efficiently process RDF data. Furthermore, column-oriented stores (Idreos et al., 2012; MacNicol and French, 2004; Stonebraker et al., 2005; Zukowski et al., 2006) exploit fully decomposed storage model (Copeland and Khoshafian, 1985) to scale-up to large datasets, and data factorization based query optimization techniques are proposed in (Bakibayev et al., 2013).

## 3.1 RDF Data Compression

A user specific approach to minimize RDF graphs by defining Datalog rules to remove the irrelevant RDF data is proposed by Meier (Meier, 2008). Similarly, Pichler et al. (Pichler et al., 2010) study the complexity of RDF minimization in presence of constraints, rules, and queries. These approaches require data decompression to process and manage the compressed RDF datasets. A graph pattern based logical compression technique, proposed by Pan et al. (Pan et al., 2014), replaces bigger graph patterns by smaller graph patterns and generates a sequence of bits for each graph pattern. Similarly, Fernandez et al. (Fernández et al., 2013) compresses and describes RDF data in binary format in terms of header, dictionary, and triples. The header contains compression relevant metadata, the dictionary contains identifiers of data values and triples represent the collection of data identifiers. A compressed RDF structure, $k^2$-triples, presented by Álvarez-García et al. (Álvarez-García et al., 2011), vertically partitions RDF triples, and utilizes $k^2$-trees (Brisaboa et al., 2009) to create indexes for each partition. Bok et al. (Bok et al., 2019) present RDF provenance compression technique by exploiting dictionary encoding. These approaches provide effective solutions for RDF data compression. However, customized engines are required to execute queries over the compressed RDF data, and data management tasks demand decompression techniques to be performed over the compressed data. Contrary, we propose factorization techniques that generate a compact representation by exploiting properties of semantic sensor data, where queries can be executed directly over the compact representations. Since factorization and compression techniques are independent, and do not directly intervene with each other, both can be exploited in conjunction.

## 3.2 Big Data Tools and RDF

Relational representations of RDF data over big data storage technologies, i.e., Parquet and MongoDB, are presented by Mami et al. (Mami et al., 2016), where a table for each RDF class is created, representing class properties as attributes. Du et al. (Du et al., 2012) combine Hadoop framework and an RDF triple store, Sesame, to achieve scalable RDF data analysis. RDF data is partitioned in such a way that all the triples with the same predicate are allocated to the same partition. Jena-HBase (Khadilkar et al., 2012) provides a variety of RDF data storage layouts for HBase and all operations over RDF graph are converted into the underlying layout operations. Schätzle et al. (Schätzle et al., 2013) present PigSPARQL, a SPARQL query processing framework using Hadoop MapReduce over large RDF graphs. A scalable RDF data management system developed by Punnoose et al. (Punnoose et al., 2012) presents storage methods and indexing, where RDF data is stored as a pair of a key and a corresponding value, and RDF triples are indexed using SPO, POS, and OSP. Papailiou et al. (Papailiou et al., 2013) present an RDF store to efficiently perform distributed Merge and Sort-Merge joins using multiple indexing over HBase, where indexes that are compressed using dictionary encoding. Nie et al. (Nie et al., 2012) study the efficient RDF partitioning and indexing schemes to process RDF data in distributed way using MapReduce. We propose factorization techniques for the RDF sensor data where the RDF triples related to the redundant values are factorized. The proposed tabular-based representations of factorized RDF graphs, i.e., factorized tables and CT based tables, scale up to large datasets by leveraging the column-oriented Parquet storage format. The tabular representation of the factorized RDF graphs remove data redundancies and improve the storage and query processing using Big Data tools.

## 3.3 Relational Data Compression

Column-stores have gained attention for being able to efficiently store data and improve the IO bandwidth for large-scale data intensive applications. Early efforts include C-Store (Stonebraker et al., 2005), SybaseIQ (MacNicol and French, 2004), MonetDB (Idreos et al., 2012), and lightweight data compression by Zukowski (Zukowski et al., 2006). Various compression techniques are exploited in C-Store (Stonebraker et al., 2005) to support several column sort-orders without space explosion. C-Store compresses each column using one of the four encoding schemes defined based on the order of values in the column. Similarly, SybaseIQ (MacNicol and French, 2004) uses column-store to perform complex analytics efficiently on massive amounts of data, and optimizes workloads across multiple servers through multi-node, shared storage, and parallel database system. MonetDB (Idreos et al., 2012) exploits column-

store technology to efficiently perform analytics over the large collections of data. Furthermore, lightweight compression is used to keep intermediate results in memory for reuse. Zukowski (Zukowski et al., 2006) proposes lightweight data compression techniques over the column-stores in order to speedup the data-intensive query processing. These compression techniques exploit column-oriented stores that use fully decomposed storage model by Copeland et al. (Copeland and Khoshafian, 1985), where *n-array* relations are decomposed into *n* binary relations, i.e., a pair of attribute value and an identifier. Two copies of each binary relation are stored increasing the storage requirements. Our approach generates a factorized RDF graph where data redundancies are reduced. Hence, factorized representations reduce the storage requirements for the decomposition storage model.

## 3.4 Data Compression based Query Optimization

Factorization techniques have been utilized for optimization of relational data and SQL query processing (Bakibayev et al., 2013; Bakibayev et al., 2012). Existing approaches proposed compact representations of relational data, obtained by applying logical axioms of relational algebra, e.g., distributivity of product over union, and commutativity of product and union. Bakibayev et al. (Bakibayev et al., 2012) present an in-memory query engine to run select-project-join queries over factorized data. The query results are expressed using factorized representations in terms of singleton relations, product, and union. The compact representations are obtained by algebraic factorization using distributivity of product over union, and the commutativity of product and union. These factorized representations form a nested structure containing the attributes from the schema, and are referred as factorization tree. A set of operators for selection and projection are proposed that map the factorized representations and generate efficient query plans. Similarly, Bakibayev et al. (Bakibayev et al., 2013) improve the performance of relational processing for aggregate and ordering queries using the distributivity of product over union to factorize relations as in the factorization of logic functions (Brayton, 1987). Factorized representations reduce the number of computations required for the evaluation of aggregation functions, i.e., sum, count, avg, min, max, likewise evaluation of aggregation functions as sequences of partial aggregations over the factorized representation speedup the processing. To evaluate order-by-queries, factorized tables are restructured with a constant delay enumeration. Therefore, queries can be executed in factorized relational data, and efficient execution plans can be found to speed up execution time. We build on these experimental results and propose factorization technique tailored for semantically described sensor data. The *CSSD* approach exploits the semantics encoded in RDF sensor data to compactly represent RDF triples, reduce redundancy, and facilitate query execution.

# 4 The Semantic Sensor Data Factorization Approach

## 4.1 Preliminaries

The Semantic Sensor Network (SSN) Ontology (Compton et al., 2012), developed by the W3C Semantic Sensor Network Incubator Group[2], is an OWL ontology that consists of 50 RDF classes and 55 properties to semantically describe sensor data in terms of observations, observed properties, features of interest, and measurement units and observed values. A portion of the SSN ontology, illustrated in Figure 2a. Sensors generate observations by detecting certain properties of features of interest and produce the observed values as sensor output. Given disjoint infinite sets $\mathbf{I}$, $\mathbf{L}$, $\mathbf{B}$ of IRIs, literals, and blank nodes, respectively, a tuple $(s\ p\ o) \in (I \cup B) \times I \times (I \cup B \cup L)$ is called an RDF triple. An RDF graph $G = (V_G, E_G, L_G)$ comprises RDF triples, where $V_G$ is a set of nodes in $\mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$, $E_G$ is a set of edges representing RDF triples, and $L_G$ is a set of edge labels in $\mathbf{I}$ (Arenas et al., 2009). Figure 3a illustrates an RDF graph representing a portion of the RDF dataset from the weather observations. Nodes correspond to resources representing sensor observations, timestamps, measurements, and literals. Edges in RDF graphs represent RDF triples and connect the nodes in RDF graphs using properties from the SSN ontology. We ignore name of the properties, prefixes, and replace long URLs by short identifiers for clarity. We refer to such an RDF graph described using the SSN ontology in this paper as an SSN RDF graph. RDF graphs are usually composed of entity description sub-graphs, sometimes also referred to as Concise Bounded Descriptions (CBD)[3]. These subgraphs are named *RDF subject molecules* defined as follows: Given an RDF graph G, a subgraph *M* of *G* is an *RDF molecule* (Fernández et al., 2014) iff the RDF triples of $M = \{t_1, \ldots, t_n\}$ share the same subject, i.e., $\forall\ i, j \in \{1, .., n\}\ (subject(t_i) = subject(t_j))$.

---

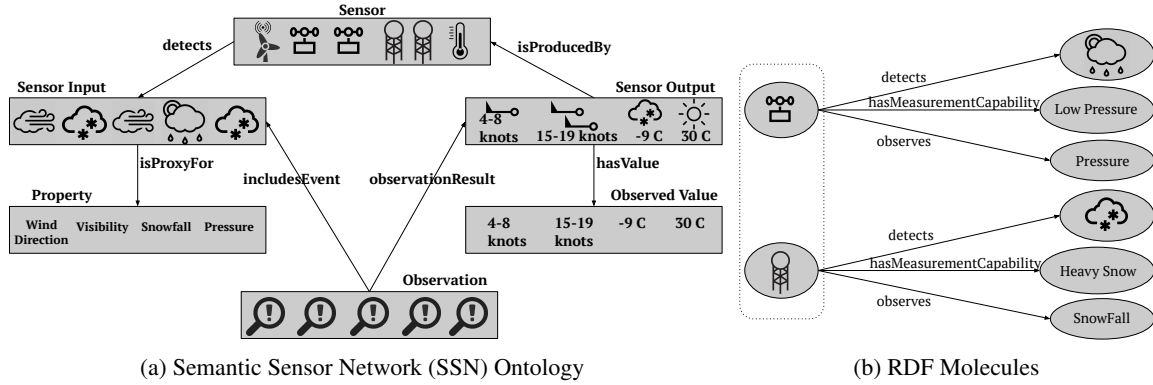(a) Semantic Sensor Network (SSN) Ontology      (b) RDF Molecules

Figure 2: **Overview of the Semantic Sensor Network (SSN) Ontology**. (a) The SSN Ontology is composed of 50 classes and 55 properties to describe sensor observations; a portion of the SSN classes and properties is presented. (b) An RDF graph with two subject molecules in the class Sensor; for clarity URIs are omitted.

Figure 2b presents an RDF graph with two RDF subject molecules. Each RDF molecule consists of three RDF triples connected to the same subject, which represents an instance of the sensor class. We will refer to RDF subject molecules as *molecules* in the rest of the paper.

## 4.2 Problem Statement

The concept of RDF molecule is utilized to devise observation and measurement molecules based on the SSN Ontology. Moreover, we present the concept of multiplicity. Building on these definitions the problem tackled in this work is defined.

**Definition 4.1** (Observation Molecule). An observation molecule *OM* is a set of RDF triples that share the same subject of type observation class, i.e., *OM=* (*obs* rdf:type *:Observation*), (*obs* :procedure *proc*), (*obs* :property *pp*).

Figure 3b presents three observation molecules, each consists of three RDF triples describing an observation subject. Each observation subject is described in terms of observation type, observed property and the observation procedure.

**Definition 4.2** (Measurement Molecule). A measurement molecule *MM* is a set of RDF triples that share the same measurement subject, i.e., *MM=* (*m* rdf:type *:MeasureData*), (*m* :value *val*), (*m* :unit *uom*).

Figure 3c presents three measurement molecules, each consists of three RDF triples having the same measurement subject. Each measurement is described in terms of measured value and unit. Class Templates are the abstract descriptions of the triples in RDF graphs and are defined as follows:

**Definition 4.3** (Class Template (CT)). Given a class *C* in an RDF graph *G*, a Class Template is a $4-$

$tuple =< C, SP, IntraL, InterL >$, where, *SP* is a set of properties in *C*, *IntraL* is a set of pairs $(p, C_j)$ such that *p* is an object property with domain *C* and range $C_j$ in the same dataset, and *InterL* is a set of pairs $(p, C_k)$ such that *p* is an object property with domain *C* and range $C_k$ in different datasets. A Class Template is a simplification of an RDF Molecule Template (Endris et al., 2017).

Figure 4 shows class templates (CT) extracted from Figure 3a around the :TempObs, :RainfallObs, :Instant, and :MeasureData classes (Figure 4a). Moreover, Figure 4b shows intra-link between :MeasureData and :TempObs using :result, and inter-link of :TempObs and :RainfallObs to :Instant using :samplingTime.

**Definition 4.4** (Measurement Multiplicity (Karim et al., 2017)). Given an RDF graph *G* of sensor data using the SSN ontology. Given a resource uom corresponding to a measurement unit, and a literal value val, the measurement multiplicity of uom and val in *G*, $M_m(val, uom|G)$, is defined as the number of measurements have same value *val* and measurement unit *uom* in *G*.

$$M_m(val, uom|G) = |\{m| (m \text{ rdf:type :MeasureData})$$
$$\in G, (m \text{ :unit } uom) \in G,$$
$$(m \text{ :value } val) \in G\}|$$

In the RDF graph in Figure 3a, three measurements, i.e., :m1, :m2, and :m3, are related to the unit cm and value 20.0. Therefore, the measurement multiplicity of cm and 20.0 is 3. Similarly, the measurement multiplicity of °F and 24.8 is 3.

**Definition 4.5** (Observation Multiplicity (Karim et al., 2017)). Given an RDF graph *G* of sensor data described using the SSN ontology. Given resources proc, ph, pp, and uom corresponding to a procedure, an observed phenomenon, observed property,

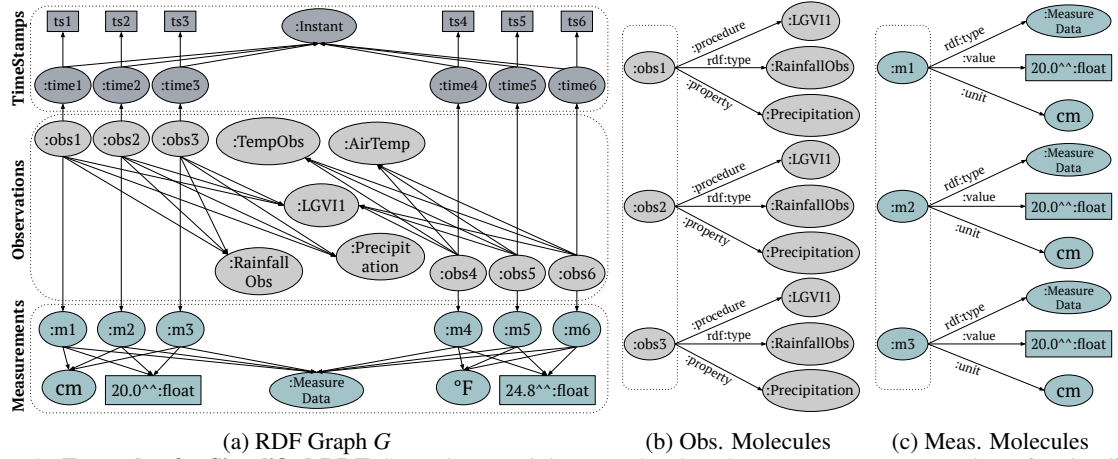(a) RDF Graph $G$      (b) Obs. Molecules      (c) Meas. Molecules

Figure 3: **Example of a Simplified RDF.** Several RDF triples are related to the same measurement values, for simplicity URIs are not presented. (a) RDF graph has $M_m(v,u|G) = 3$ and $M_o(s,p,pp,v,u|G) = 3$ for the values $20.0cm$ and $24.8°F$. (b) Three observation (Obs.) molecules; (c) Three measurement (Meas.) molecules.



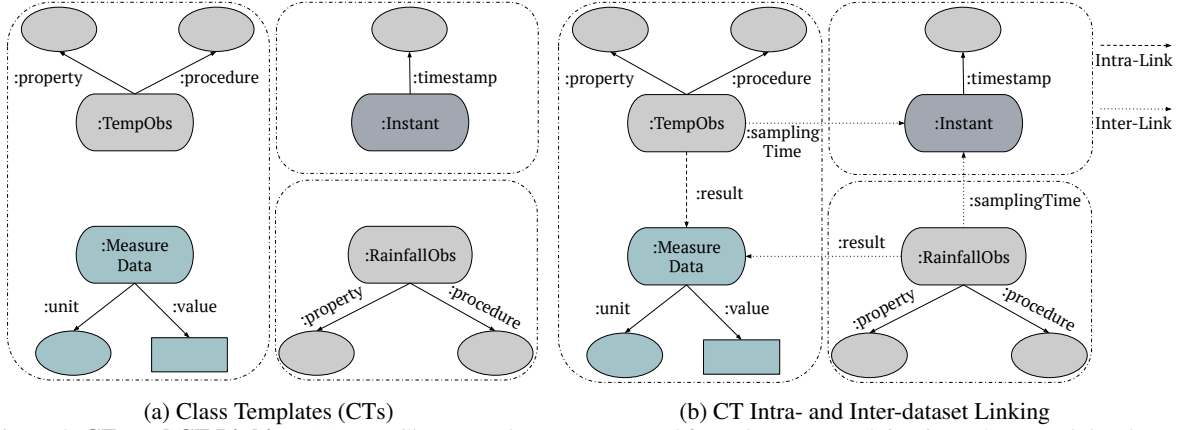(a) Class Templates (CTs)      (b) CT Intra- and Inter-dataset Linking

Figure 4: **CTs and CT Linking**. (a) Four Class Templates are extracted from the RDF graph in Figure 3a around the classes `:TempObs`, `:RainfallObs`, `:MeasureData`, and `:Instant`; (b) inter- and intra-links between class templates.

and measurement unit, and a literal value `val`, the multiplicity of an observation `obs` for `uom` and `val` in $G$, $M_o(proc, ph, pp, val, uom|G)$, is defined as the number of observations about the property `pp` of the observed phenomenon `ph`, sensed by `proc`, that have the same value `val` and unit of measurement `uom` in $G$.

$$M_o(proc, ph, pp, val, uom|G) = |\{obs|$$
$$(obs \text{ rdf:type } ph)$$
$$\in G, (obs \text{ :procedure}$$
$$proc) \in G, (obs \text{ :property } pp) \in G, (obs$$
$$\text{:result } m) \in G, (m$$
$$\text{rdf:type :Measure}$$
$$\text{Data}) \in G, (m \text{ :unit}$$
$$uom) \in G, (m \text{ :value}$$
$$val) \in G\}|$$

In Figure 3a, the procedure `:LGVI1`, the phenomenon `:RainfallObs`, the property

`:Precipitation`, the measurement unit `cm`, and the value `20.0` are associated with `:obs1`, `:obs2`, and `:obs3`, and the observation multiplicity is 3. Similarly, the observation multiplicity for `:LGVI1`, `:TempObs`, `:AirTemp`, `°F`, and `24.8` is 3.

**Definition 4.6** (Compact Observation Molecule). Given a surrogate observation *oM*, a compact observation molecule *COM* is a set of RDF triples that share the same surrogate observation *oM*, i.e., *COM*= (*oM* rdf:type *:Observation*),(*oM* :procedure *proc*), (*oM* :property *pp*), such that the observation multiplicity of the procedure `proc` and the observed property `pp` is one.

Figure 5a presents a compact observation molecule, with a surrogate observation `:obsM1` that corresponds to the observations `:obs1`, `:obs2`, and `:obs3` in Figure 3b, and is associated with the observation multiplicity value one.

**Definition 4.7** (Compact Measurement Molecule). Given a surrogate measurement *mM*, a compact measurement molecule *CMM* is a set of RDF triples that share the same surrogate measurement *mM*, i.e., *CMM*= (*mM* rdf:type *:MeasureData*), (*mM* :value *val*), (*mM* :unit *uom*), such that the multiplicity of the value val and the unit uom is one.

A compact measurement molecule for the measurement molecules in Figure 3c is presented in Figure 5a using a surrogate measurement :mM1. :mM1 corresponds to :m1, :m2, and :m3 in Figure 3c and is associated with the multiplicity value one.

**Definition 4.8** (A Factorized RDF Graph). Given an RDF graph $G = (V_G, E_G, L_G)$ representing sensor data described using the SSN ontology, a factorized RDF graph $G' = (V_{G'}, E_{G'}, L_{G'})$ of $G$ is an RDF graph where the following hold:

- Entities in $G$ are preserved in $G'$, i.e., $V_G \subseteq V_{G'}$.
- For each entity *obs* in $V_G$ that corresponds to an entity of :Observation class over the class properties :procedure and :property and objects *proc* and *pp*, respectively, in $G$, there is an entity *oM* in $V_{G'}$ that corresponds to the surrogate observation of the compact observation molecule over the properties :procedure and :property and objects *proc* and *pp*, respectively, in $G'$.
- For each entity *m* in $V_G$ that corresponds to an entity of class :MeasureData over the properties :value and :unit and objects *val* and *uom*, respectively, in $G$, there is an entity *mM* in $V_{G'}$ that corresponds to the surrogate measurement of the compact measurement molecule over the properties :value and :unit and objects *val* and *uom*, respectively, in $G'$.
- There is a partial mapping $\mu_N$: $V_G \rightarrow V_{G'}$:
  - Observation entities in $G$ are mapped to the surrogate observations in $G'$, i.e., $\mu_N(obs)=oM$.
  - Measurement entities in $G$ are mapped to the surrogate measurements in $G'$, i.e., $\mu_N(m)=mM$.
  - The mapping $\mu_N$ is not defined for the rest of the entities that are not instances of the :Observation or :MeasureData class in $G$.
- For each RDF triple *t* in (*s p o*) in $E_G$:
  - If $\mu_N(s)$ is defined and :Observation is the type of *s*, then the RDF triples (*s* :observationOf $\mu_N(s)$) and ($\mu_N(s)$ rdf:type :Observation) is in $E_{G'}$.
  - If $\mu_N(s)$ is defined and :MeasureData is the type of *s*, then the RDF triple ($\mu_N(s)$ rdf:type :MeasureData) belong to $E_{G'}$.

- If $\mu_N(s)$ is defined and :Observation is the type of *s*, and *p* is not :result and :samplingTime, then ($\mu_N(s)$ *p o*) is in $E_{G'}$.
- If *p* is :samplingTime, then (*s p o*) is in $E_{G'}$.
- If $\mu_N(s)$ and $\mu_N(o)$ are defined and *p* is :result, then ($\mu_N(s)$ *p* $\mu_N(o)$)) and (*s p o*) are in $E_{G'}$.
- If $\mu_N(s)$ is defined and type of *s* is :MeasureData, then ($\mu_N(s)$ *p o*) is in $E_{G'}$.
- Otherwise, the RDF triple *t* is preserved in $E_{G'}$.
- Multiplicity of measurements is reduced, i.e., for all *val*, *uom* such that $M_m(val, uom|G) \geq 1$, then $M_m(val, uom|G')$=1, and
- Multiplicity of observations is reduced, i.e., for all *proc*, *ph*, *pp*, *val*, *uom*, such that, $M_o(proc, ph, pp, val, uom|G) \geq 1$, then $M_o(proc, ph, pp, val, uom|G')$=1.

Given an RDF graph $G = (V_G, E_G, L_G)$ representing sensor data with the SSN ontology, the problem of *semantic sensor data factorization (SSDF)* in $G$, corresponds to finding a *factorized RDF graph* $G' = (V_{G'}, E_{G'}, L_{G'})$ of $G$. Consider RDF graphs $G$ and $G'$ in Figure 3a and Figure 5c, respectively. Furthermore, Figure 5b shows mappings $\mu_N$ that assign measurements :m1, :m2, and :m3 in $G$ to the surrogate measurement :mM1 in $G'$, and :m4, :m5, and :m6 to :mM2. Similarly, :obs1, :obs2, and :obs3 are mapped to :obsM1, and :obs4, :obs5, and :obs6 to :obsM2; $\mu_N$ is the identity for the rest of the nodes. Measurement and observation multiplicities are one in $G'$, which is the *factorized graph* of $G$.

Once RDF graphs are factorized, query processing is performed against the factorized graphs. SPARQL queries over the original RDF graphs need to be re-written against the corresponding factorized RDF graphs in the way that equivalent answers are computed. We have defined seven query rewriting rules, given in Table 1. Each rule is given a name, i.e., *fssn1, fssn2, fssn3, fssn4, fssn5, fssn6,* and *fssn7,* and has a head and a body. The head of a rule corresponds to the triple pattern in the query against original RDF graph, whereas the body of the rule represents the corresponding triple patterns against the factorized RDF graph. The head of the rule *fssn1* contains a triple pattern that matches to all the original observations, whereas the body of the rule matches the corresponding surrogate observations. Moreover, the variable substitutions, i.e., *?obs* by *?Xobs*, are maintained for the query clauses such as SELECT, FILTER, GROUP BY etc. The head of rule *fssn2*, matches the procedure generating the original observations, while the body of the rule matches the procedure of the surrogate observations, and keeps the variable substitutions. Similarly, the head of the rule *fssn3* consists of a triple

Table 1: **Query Rewriting Rules**. The rewriting rules for observations and measurements with respect to the relevant properties are expressed in terms of triple patterns. The variables representing observations and measurements are replaced in SPARQL query clauses, i.e., SELECT, ORDER BY, GROUP BY, and FILTER.

| Rule Name | Head | Body |
|---|---|---|
| fssn1 | *?obs rdf:type :Observation* | *?obs rdf:type :Observation*<br>*?Xobs :observationOf ?obs*<br>Replace *?obs* by *?Xobs* in query clauses |
| fssn2 | *?obs :procedure ?sensor* | *?obs :procedure ?sensor*<br>*?Xobs :observationOf ?obs*<br>Replace *?obs* by *?Xobs* in query clauses |
| fssn3 | *?obs :property ?property* | *?obs :property ?property*<br>*?Xobs :observationOf ?obs*<br>Replace *?obs* by *?Xobs* in query clauses |
| fssn4 | *?m rdf:type :MeasureData* | *?m rdf:type :MeasureData*<br>*?Xobs :observationOf ?obs*<br>*?Xobs :result ?Xm*<br>Replace *?m* by *?Xm* in query clauses |
| fssn5 | *?m :value ?val* | *?m :value ?val*<br>*?Xobs :observationOf ?obs*<br>*?Xobs :result ?Xm*<br>Replace *?m* by *?Xm* in query clauses |
| fssn6 | *?m :unit ?uom* | *?m :unit ?uom*<br>*?Xobs :observationOf ?obs*<br>*?Xobs :result ?Xm*<br>Replace *?m* by *?Xm* in query clauses |
| fssn7 | *?obs :result ?m* | *?obs :result ?m*<br>*?Xobs :observationOf ?obs*<br>*?Xobs :result ?Xm*<br>Replace *?obs* by *?Xobs* and *?m* by *?Xm* in query clauses |

(a) Compact Molecules     (b) Entity mappings $\mu_N$ from $G$ into $G'$
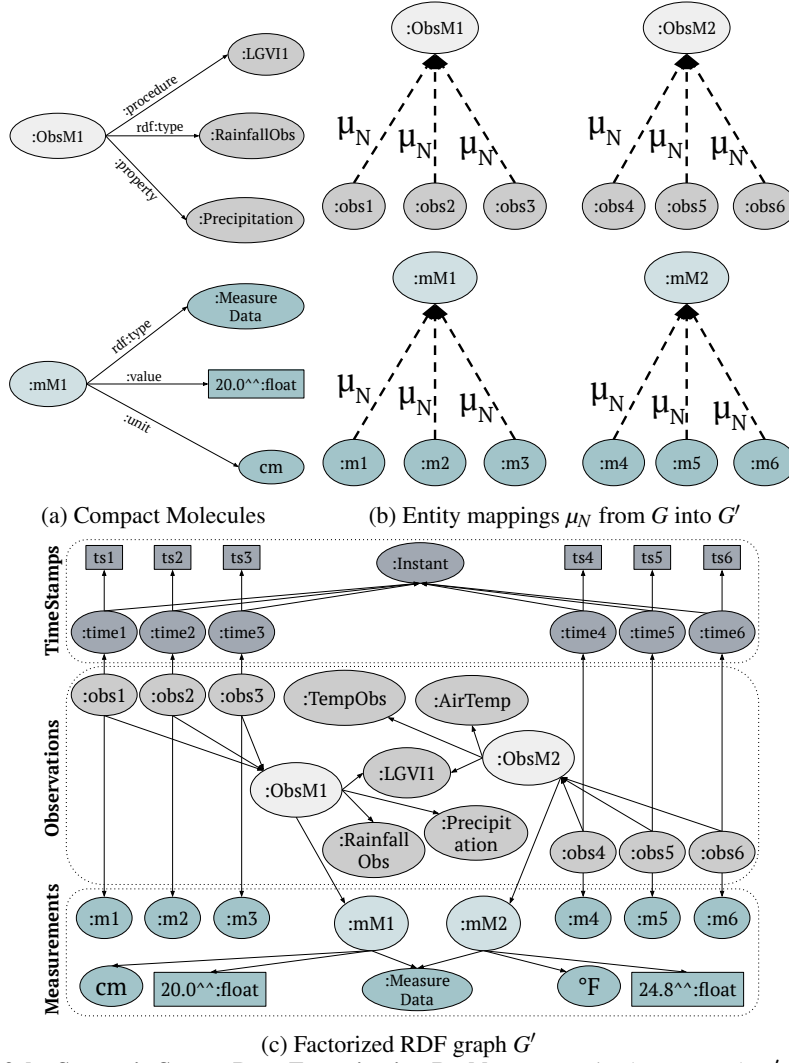
(c) Factorized RDF graph $G'$

Figure 5: **Instance of the Semantic Sensor Data Factorization Problem.** Factorized RDF graph $G'$ of $G$ in Figure 3a. (a) Compact observation and measurement molecules are presented. (b) Entity mappings $\mu_N$ from graph in Figure 3a to surrogate entities; (c) Factorized RDF graph $G'$ with multiplicities equal to one.

pattern that matches the observed property in the original RDF graph, and the body of the rule extracts the observed property of the surrogate observations.

The rules *fssn4*, *fssn5*, and *fssn6* are used to rewrite the triple patterns involving the measurement properties. The head of the rule *fssn4* contains a triple pattern matching all the entities of measurements in original RDF graphs. The body of the rule *fssn4* contains three triple patterns that match to the surrogate measurements, as well as, associate the original observations with the surrogate observations, using property *:observationOf*, and relate observations to corresponding measurements in the original RDF graph using property *:result*. Moreover, the body of the rule maintains the measurement variable substitutions. The head of the rule *fssn5* find matches of

the values of measurements in original RDF graphs, whereas the body of the rule find values of the surrogate measurements in factorized RDF graphs. Further, the triple patterns extract associations between the original and surrogate observations, as well as between the original observations and corresponding original measurements, and maintain the measurement variable substitutions. The head of rule *fssn6* contains the triple pattern matching the measurement units in original RDF graph, whereas the body matches the unit of the surrogate measurements. Also, body maintains associations between original and surrogate observations and original observations and corresponding measurements along with measurement variable substitutions. Finally, the head of the rule *fssn7* maps the original observations and the

measurements in original RDF graphs using property *fssn7*. The body of the rule *fssn7* find associations between surrogate observations and surrogate measurements in factorized RDF graphs using property *:result*. Likewise associations between the original and surrogate observations and the original observations and original measurements are maintained. Additionally, the variable substitutions for the observations and measurements are maintained.

Let $G$ and $G'$ be RDF graphs such that $G'$ is a factorized graph of $G$. Consider a SPARQL query $Q$ over $G$. The problem of *evaluating SPARQL queries against a factorized RDF graph* corresponds to the problem of transforming $Q$ into a SPARQL query $Q'$ over $G'$ such that the results of evaluating $Q$ over $G$ and the results of $Q'$ over $G'$ are the same, i.e., the condition $[[Q]]_G = [[Q']]_{G'}$ is satisfied.

An instance of the problem of evaluating queries on factorized RDF graphs is shown in Figure 6. A SPARQL query $Q$ over the RDF graph $G$ in Figure 3a is presented in Figure 6a. The SPARQL query $Q'$ in Figure 6b, corresponds to a rewriting of $Q$, against $G'$ which represents factorization of $G$. The evaluations of $Q$ and $Q'$ produce the same answers. In this work, we present SPARQL query rewriting rules that allow for rewriting a query $Q$ into a query $Q'$.

## 4.3 A Factorization Approach

We present a solution to the *semantic sensor data factorization (SSDF)* problem. A sketch of the proposed factorization approach is presented in Algorithm 1. The algorithm receives an RDF graph $G(V_G, E_G, L_G)$, and a graph $G''(V_{G''}, E_{G''}, L_{G''})$ representing an already factorized graph and the entity mappings $\mu_{N''}$ in $G''$. The algorithm incrementally generates a factorized RDF graph $G'(V_{G'}, E_{G'}, L_{G'})$ of $G$, and the entity mappings $\mu_N$ from the observations and measurements in $G$ to the surrogate observations and measurements in $G'$, respectively. The algorithm initializes a set of entity mappings $\mu_N$ and the sets of nodes $V_{G'}$, edges $E_{G'}$, and labels $L_{G'}$ of the factorized graph $G'$ (line 1). If for all the observations with observed phenomenon *ph*, sensor procedure *proc*, observed property *pp*, and for the corresponding measurements with value *val* and the related unit *uom* in $G$, the surrogate observation and measurement are already in $G''$, then the observations and measurements in $G$ are mapped in $\mu_N$ to the surrogate observation and measurement in $G''$, respectively(lines 2-9). Furthermore, observations in $G$ are linked using the property `:observationOf` to the surrogate observations in $\mu_N$(line 6-8). If the surrogate observation and measurement are not in $G''$, the algorithm (lines
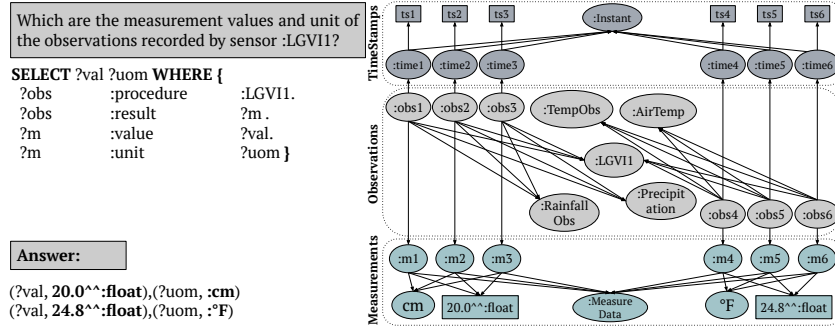
11) creates corresponding surrogate entities in $G'$, i.e., the subjects of compact measurement and observation molecules are created. In lines 12-13, the algorithm maps all the measurements, related to *val* and *uom* in $G$, to the surrogate measurements in $\mu_N$. For all the observations with observed phenomenon *ph*, sensor procedure *proc*, observed property *pp*, measurement value *val* and unit of measurement *uom* in $G$, adds in $\mu_N$ the mappings of all the observations in $G$ with the surrogate observations in $G'$ in lines 14-15. Once the mappings are in $\mu_N$, the nodes and edges representing the mapped observations and measurements in $G$ are processed. All nodes $s$ and $o$ related to the property `:result` in $G$ are added to $G'$ along with their associations. Moreover, a new edge relating $s$ and $\mu_N(s)$ using the property `:observationOf` is added to $G'$ (lines 18-20). If $s$ and $o$ are linked using a property `rdf:type` and $o$ is either `:Observation` or `:MeasureData`, then a new edge $(\mu_N(s)\ p\ o)$ is added to $G'$ along with $\mu_N(s)$ and $o$ (lines 21-22). If $s$ and $o$ are associated through a predicate $p$ in {:procedure, :property, :value, :unit}, then a new edge $(\mu_N(s)\ p\ o)$ is added to $G'$ in lines 23-24. Otherwise, the edge $(s\ p\ o)$ is added to the $G'$ in lines 25-26.

Figure 7b depicts a portion of the RDF in Figure 3a and the corresponding transformation in the factorized RDF graph in Figure 5c. The surrogate measurements and observations, and the new edges are highlighted in bold. The Algorithm 1 creates the surrogate measurements and observations in line 3 and 7; new edges are created in line 12, 15, and 17. Additionally, assumptions about the characteristics of the associations between the nodes in the graph are presented. While some edges existing in the RDF graph in Figure 3a are not present in the factorized RDF graph, these associations can be obtained by traversing the graph through the surrogate observations and measurements. The implicit satisfaction of all the associations in the original RDF graph that are not included in the factorized graph is restricted under the following assumptions:

For all observations `:obs` and measurements `:m` in $G$, the following hold.

- **Measurement**: the properties `:value` and `:unit` of measurement are both functional properties for any measurement `:m`. Furthermore, the property `:result` that associates an observation with a measurement has a functional inverse.

- **Observation**: the property `:procedure` that associates an observation and a procedure is a functional property for any observation `:obs`.

The following hold for a surrogate observation `:obsM`, a surrogate measurement `:mM`, an observation `:obs`, and a measurement `:m` in $G'$.

(a) SPARQL Query over Original RDF Graph



(b) SPARQL Query over Factorized RDF Graph

Figure 6: **Instance of the Query Evaluation Problem**. Evaluation of SPARQL queries over the original and factorized RDF graphs respects set semantics. (a) SPARQL query over original RDF graph selects the values and unit collected by *:LGVI1*; (b) Rewritten SPARQL query over factorized RDF graph, in Figure 5c.

- **Surrogate Observation**: The properties `:procedure` and `:property` are functional properties for any surrogate observation `:obsM`.

- **Surrogate Measurement**: `:value` and `:unit` are both functional properties for any surrogate measurement `:mM`. Furthermore, `:result` that associates a surrogate observation with a surrogate measurement has a functional inverse.

- **Observation**: `:observationOf` property that associates an observation `:obs` with a surrogate observation is a functional property.

- **Measurement**: `:m` is related to only one observation, i.e., `:result` associates an observation with a measurement, and has a functional inverse.
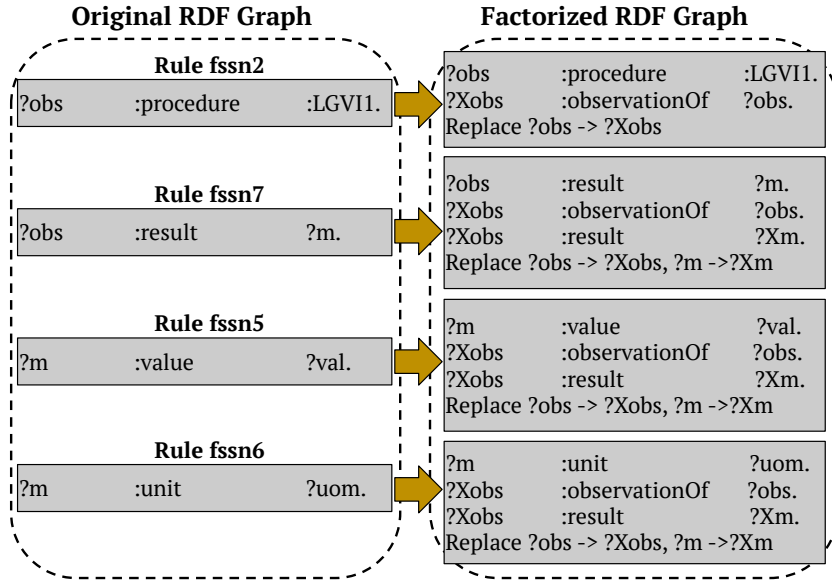
We are assuming that SPARQL queries against the original and factorized RDF graphs are evaluated under the set semantics, i.e., no duplicates are in the answers. Coming back to the motivating example, Figure 8 illustrates the factorized RDF graph of the graph in Figure 1. The factorized RDF graph in Figure 8a is sparse and the average number of neighbors has been reduced from 6.4 to 2.5. This indicates that the number of RDF triples describing an observation is reduced after factorization. Figure 8c shows that for each measurement value the number of associated

RDF triples in the factorized RDF graph is reduced by 74%.

## 4.4 Queries over Factorized RDF Graphs

In this section, we define the algorithm that solves the problem of query evaluation on a factorized RDF graph. Table 1 presents the rules to rewrite a SPARQL query against an original SSN RDF graph into a query against the corresponding factorized RDF graph. The query rewriting rules are defined in terms of SPARQL triple patterns. For each property of the observation and measurement classes, a rewriting rule is defined. Furthermore, substitutions for the observation and measurement variables in the query clauses, i.e., SELECT, ORDER BY, GROUP BY, and FILTERS etc, are presented. Given a SPARQL query and a set *R* of query rewriting rules, the Algorithm 2 describes the steps performed to each set of triple patterns that composes a Basic Graph Pattern (BGP). If the input query consists of several BGPs, the structure of the original query remains the same, and Algorithm 2 is applied to each BGP within the query using rules in Table 1.

Figure 6 presents two SPARQL queries: Figure 6a

(a) Query Rewriting

(b) Original and Factorized RDF Graphs

Figure 7: **Example of Query Rewriting**. Query rewriting rules are presented. (a) Query rewriting rules from Table 1 are used to rewrite the query in Figure 6a into the query in Figure 6b. (b) Portions of the RDF graphs (original and factorized). Nodes and edges highlighted in bold are added during the RDF graph factorization.

and Figure 6b present an original query *Q* and rewriting of *Q* produced by Algorithm 2. Figure 7a shows the rewriting of SPARQL query in Figure 6a. Rules *fssn2*, *fssn5*, *fssn6*, and *fssn7* from Table 1 are used to rewrite the query. The algorithm replaces each triple pattern in a BGP that instantiates the head of a rule in *SR* by the body of the rule, e.g., the triple pattern (*?obs :procedure :LGVI1*) instantiates the head of rule *fssn2*, thus, the triple pattern in the BGP is replaced with the body of *fssn2*, as shown in Figure 7a. Moreover, the variables corresponding to the observations

and measurements in the original query represent the surrogate observations and measurements in the rewritten query, consequently, these variables are replaced by the new variables in the query clauses. The variable substitution for observation *?obs* by *?Xobs* is maintained during the rewriting process using rule *fssn2* in order to retrieve the original observations, if required. Similarly, other triple patterns in the BGP each matching the head of a rule, i.e., *fssn5*, *fssn6*, and *fssn7*, are replaced by the body of the rule, and the variable substitutions of *?obs* and *?m* by *?Xobs*

| S# | Parameter | Value |
|----|-----------|-------|
| 1 | Connected Components | 1.0 |
| 2 | Network Centralization | 0.1 |
| **3** | **Avg. # of Neighbors** | **2.5** |
| 4 | Network Density | 0.0 |
| 5 | Multi-edge Node Pairs | 5.0 |
| 6 | Network Heterogeneity | 9.2 |

(a) Fact. **RDF Graph**    (b) **Statistics** of Factorized RDF Graph
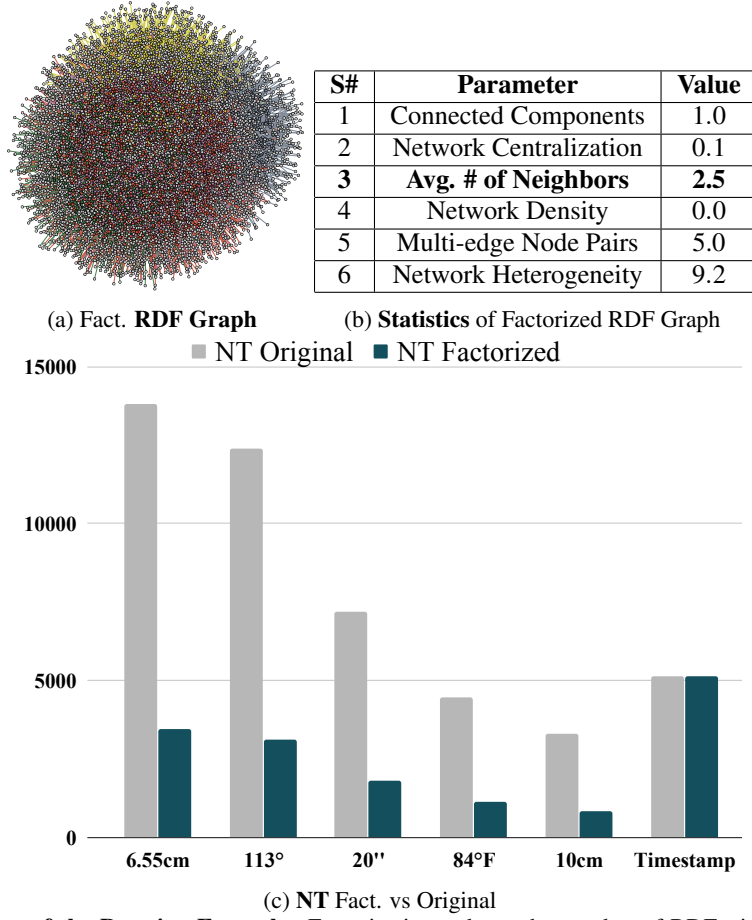


(c) **NT** Fact. vs Original

Figure 8: **Factorization of the Running Example.** Factorization reduces the number of RDF triples related to the same value. (a) Factorized (Fact.) RDF Graph of Figure 1a; (b) Statistics of the fact. RDF graph; (c) The number of factorized triples. The graph and statistics are generated by the `Cytoscape tool` (`http://www.cytoscape.org/`).

and $?Xm$, respectively, are maintained for the query clauses. The evaluation of both, original and rewritten, queries produce the same results. Another important property is that the time complexity of the original and rewritten queries is also the same.

**Theorem 1.** *Given $G$ and $G'$ such that $G'$ is a factorized RDF graph of $G$. Let $Q$ and $Q'$ be SPARQL queries where $Q'$ is a rewritten query of $Q$ over $G'$ generated by Algorithm 2. The problem of evaluating $Q'$ against $G'$ is in: (1) PTIME if query $Q$ has only AND and FILTER operators; (2) NP-complete if query $Q$ has expressions with AND, FILTER, and UNION operators; and (3) PSPACE-complete for OPTIONAL graph pattern expressions.*

*Proof.* We proceed with a proof by contradiction. Assume that complexity of $Q'$ is higher than $Q$. Then, UNION or OPTIONAL operators not included in $Q$ are added to $Q'$. However, Algorithm 2 only changes triple patterns over $G$ by triple patterns against $G'$. Additionally, Algorithm 2 includes new JOINs (AND operator). However, adding AND or FILTER operators does not affect the complexity of the problem of evaluating $Q'$ over $G'$, and contradicting the fact that the complexity of $Q'$ is higher than $Q$. $\square$

# 5 Tabular Representation of RDF Graphs

Sensor data tend to stack up quickly, scaling up to large amounts of data. In order to capture that growth, we opt for representing factorized data in tabular format, so that Big Data processing technologies can be used. For that purpose, we choose to store the data in the modern, columnar-oriented *Parquet*[4] storage format. We propose tabular representations of both the original and factorized RDF

---

[4]`https://parquet.apache.org/`

**⊙Observation Universal**

| ObsID | Type | Procedure | Property | Sampling Time | Time stamp | MID | Value | Unit |
|-------|------|-----------|----------|---------------|------------|-----|-------|------|
| :obs1 | :Rainfall | :LGVI1 | :Precipitation | :time1 | ts1 | :m1 | 20.0 | cm |
| :obs2 | :Rainfall | :LGVI1 | :Precipitation | :time2 | ts2 | :m2 | 20.0 | cm |
| :obs3 | :Rainfall | :LGVI1 | :Precipitation | :time3 | ts3 | :m3 | 20.0 | cm |
| :obs4 | :Temp | :LGVI1 | :AirTemp | :time4 | ts4 | :m4 | 24.8 | °F |
| :obs5 | :Temp | :LGVI1 | :AirTemp | :time5 | ts5 | :m5 | 24.8 | °F |
| :obs6 | :Temp | :LGVI1 | :AirTemp | :time6 | ts6 | :m6 | 24.8 | °F |

(a) Universal Parquet Table for Observations

**⊙ Observation**

| ObsID | Sampling Time | Time stamp | MID | ObsMID |
|-------|---------------|------------|-----|--------|
| :obs1 | :time1 | ts1 | :m1 | :obsM1 |
| :obs2 | :time2 | ts2 | :m2 | :obsM1 |
| :obs3 | :time3 | ts3 | :m3 | :obsM1 |
| :obs4 | :time4 | ts4 | :m4 | :obsM2 |
| :obs5 | :time5 | ts5 | :m5 | :obsM2 |
| :obs6 | :time6 | ts6 | :m6 | :obsM2 |

**Compact Observation Molecule**

| ObsMID | Type | Procedure | Property | MMID |
|--------|------|-----------|----------|------|
| :obsM1 | :Rainfall | :LGVI1 | :Precipitation | :mM1 |
| :obsM2 | :Temp | :LGVI1 | :AirTemp | :mM2 |

**Compact Measurement Molecule**

| MMID | Value | Unit |
|------|-------|------|
| :mM1 | 20.0 | cm |
| :mM2 | 24.8 | °F |

(b) Factorized Data Parquet Tables

Figure 9: **Factorized Tabular Representation of RDF Graphs**. Parquet tables are utilized to represent RDF graphs in Spark. (a) A universal table stores all the data of the original RDF graph.(b) Factorized data is represented in three parquet tables to store compact observation and measurement molecules.

graphs (in Figure 3a and Figure 5c, respectively), shown in Figure 9 and Figure 11. Columnar nature of Parquet makes it best suited for scenarios where queries access only a few number of columns from a *wide* table of many columns. Parquet pulls only the requested columns, contrary to row-oriented storage. We rely on these properties of Parquet tables, and represent RDF graphs using a *universal* table. The universal tabular representation, `Observation Universal` in Figure 9a, of original RDF graph in Figure 3a, contains all the properties of an observation, i.e., `rdf:type, :procedure, :property, :result, :samplingTime, :value, :unit,` and `:timestamp`. These predicates are modeled with the attributes: `Type, Procedure, Property, MID, SamplingTime, Value, Unit,` and `Timestamp`, respectively. The tabular representation of the factorized RDF graph in Figure 5c is shown in Figure 9b. The `Compact Observation Molecule` table models the properties `rdf:type, :procedure, :property,` and `:result` of a surrogate observation with the attributes `Type, Procedure, Property,` and `MMID`, respectively. The `Compact Measurement Molecule` table contains the properties `:value` and `:unit` describing a surrogate measurement. Note that the type `:MeasureData` is implicitly represented in the ta-

ble name. The `Observation` factorized table contains the observation predicates that are not represented in the `Compact Observation Molecule` and `Compact Measurement Molecule` tables, as well as a reference to the corresponding surrogate observations, as a foreign key. Furthermore, SPARQL queries against original and factorized graphs are translated into SQL queries over universal and factorized tables, respectively. Figure 10 shows SQL representations of SPARQL queries in Figure 6. The evaluation of the SQL queries against the universal and factorized tables is the same as the SPARQL queries over the RDF graphs. Instead of using the universal tabular representations, RDF graphs can be represented using the Class Template (CT) based tabular representations. For each CT around a class one table is created containing the properties of the class as attributes. Similarly, for each intra- or inter-link between the classes a binary table is created containing the identifiers from the corresponding CT tables. Figure 11a illustrates the CT-based tabular representations around the `:RainfallObs`, `:TempObs, :Instant,` and `:MeasureData` classes in Figure 3a. The class templates of `:RainfallObs` and `:TempObs` are represented in `Rainfall CT` and `Temperature CT` with the attributes `Procedure`

**SPARQL Query**

```
SELECT ?val ?uom
WHERE {
?obs    :procedure  :LGVI1.
?obs    :result     ?m .
?m      :value      ?val.
?m      :unit       ?uom }
```

**SQL Query**

```
SELECT DISTINCT Value, Unit
FROM Observation
WHERE
    Procedure = :LGVI1
```

(a) Query Universal Table

**SPARQL Query**

```
SELECT ?val ?uom
WHERE {
?obs    :observationOf  ?oM.
?oM     :procedure      :LGVI1.
?oM     :result         ?mM.
?mM     :value          ?val.
?mM     :unit           ?uom }
```

**SQL Query**

```
SELECT DISTINCT CMM.Value, CMM.Unit
FROM Compact Observation Molecule as COM,
     Compact Measurement Molecule as CMM
WHERE
    COM.MMID=CMM.MMID AND
    COM.Procedure=:LGVI1
```
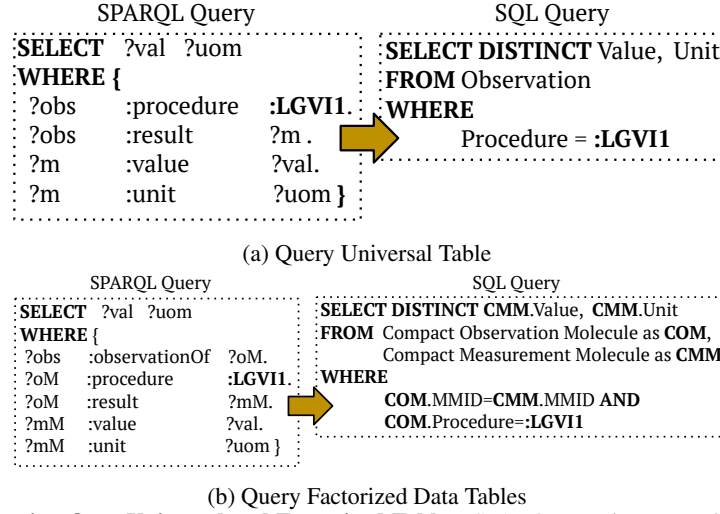
(b) Query Factorized Data Tables

Figure 10: **Query Evaluation Over Universal and Factorized Tables.** SPARQL queries over original and factorized RDF graphs and their corresponding SQL queries are presented. (a) SQL query over the universal parquet table; (b) SQL query against the parquet tables representing the factorized RDF graph.

and `Property`. Similarly, `:MeasureData` and the properties `:value` and `:unit` are represented in `Measurement CT` with the attributes `Value` and `Unit`, respectively. `Instant CT` represents `:Instant` by modeling `:timestamp` property as `Timestamp`. `Rainfall Measurement` models the association between the `:RainfallObs` and `:MeasureData` using the primary keys, `ObsID` and `MID`, from the corresponding CT-based tabular representations. Also, the association between `:RainfallObs` and `:Instant` is presented in `Rainfall Instant`. Similarly, association of `:TempObs` with `:MeasureData` and `:Instant` is presented in `Temperature Measurement` and `Temperature Instant`, respectively.

The CT-based tabular representations of the factorized RDF graph, in Figure 5c, are shown in Figure 11b. `F-Rainfall CT` models the properties `:procedure` and `:property`, describing the surrogate rainfall observations, with the attributes `Procedure` and `Property`, respectively. Similarly, the CTs of the surrogate temperature observations are modeled in `F-Temperature CT` with attributes `Procedure` and `Property`. The surrogate measurements are modeled in the `F-Measurement CT` using `Value` and `Unit`. `Instant CT` models `:timestamp` property of `:Instant` with `Timestamp`. The links between the surrogate observations and measurements are represented in `Factorized Rainfall Measurement` and `Factorized Temperature Measurement`. Moreover, the explicit mappings between the original and surrogate rainfall observations are represented in `Rainfall Observation`. Similarly, `Temperature Observation` stores the mappings between the orig-

inal and surrogate temperature observations. In addition, `Rainfall Measurement` and `Temperature Measurement` represent association of the original rainfall and temperature observations, respectively, with the corresponding measurements. Furthermore, the links of the original rainfall and temperature observations with the corresponding timestamps are represented in `Rainfall Instant` and `Temperature Instant`, respectively. Figure 12 illustrates the CT based SQL representations of the SPARQL queries in Figure 6. The results of the SQL queries against CT based tabular representations of the original and factorized RDF graphs are the same as the SPARQL queries over the original and factorized RDF graphs.

**Theorem 2.** *The decomposition of the* `Observation` *universal table into factorized tables:* `Observation`, `Compact Observation Molecule`, *and* `Compact Measurement Mole- cule`, *is* loss-less join.

*Proof.* Considering the following functional dependencies hold in the universal and factorized tables:

- `ObsMID` → `Type`, `Procedure`, `Property`, `MMID`
- `MMID` → `Value`, `Unit`
- `ObsID` → `SamplingTime`, `Timestamp`, `MID`, `ObsMID`

We can prove using the algorithm(Jeffrey, 1989) that the factorized tables are a *loss-less join* decomposition of universal table $T$ that includes all the attributes in the `Observation` universal plus `ObsMID` and `MMID`. The attributes of the `Observation` universal can be projected from $G'$, thus, satisfying the *loss-less join* condition. □

**Rainfall RDF-CT**

| ObsID | Procedure | Observerd Property |
|---|---|---|
| :obs1 | :LGVI1 | :Precipitation |
| :obs2 | :LGVI1 | :Precipitation |
| :obs3 | :LGVI1 | :Precipitation |

**Temperature RDF-CT**

| ObsID | Procedure | Observed Property |
|---|---|---|
| :obs4 | :LGVI1 | :AirTemp |
| :obs5 | :LGVI1 | :AirTemp |
| :obs6 | :LGVI1 | :AirTemp |

**Rainfall Measurement**

| ObsID | MID |
|---|---|
| :obs1 | :m1 |
| :obs2 | :m2 |
| :obs3 | :m3 |

**Temperature Measurement**

| ObsID | MID |
|---|---|
| :obs4 | :m4 |
| :obs5 | :m5 |
| :obs6 | :m6 |

**Rainfall Instant**

| ObsID | Date Time |
|---|---|
| :obs1 | :time1 |
| :obs2 | :time2 |
| :obs3 | :time3 |

**Temperature Instant**

| ObsID | Date Time |
|---|---|
| :obs4 | :time4 |
| :obs5 | :time5 |
| :obs6 | :time6 |

**Measurement RDF-CT**

| MID | Value | Unit |
|---|---|---|
| :m1 | 20.0 | cm |
| :m2 | 20.0 | cm |
| :m3 | 20.0 | cm |
| :m4 | 24.8 | °F |
| :m5 | 24.8 | °F |
| :m6 | 24.8 | °F |

**Instant RDF-CT**

| DateTime | Sampling Time |
|---|---|
| :time1 | ts1 |
| :time2 | ts2 |
| :time3 | ts3 |
| :time4 | ts4 |
| :time5 | ts5 |
| :time6 | ts6 |

(a) CT Based Parquet Table for Observations

**F-Rainfall RDF-CT**

| ObsMID | Procedure | Observed Property |
|---|---|---|
| :obsM1 | :LGVI1 | :Precipitation |

**F-Temperature RDF-CT**

| ObsMID | Procedure | Observed Property |
|---|---|---|
| :obsM2 | :LGVI1 | :AirTemp |

**Factorized Rainfall Measurement**

| ObsMID | MMID |
|---|---|
| :obsM1 | :mM1 |

**Factorized Temperature Measurement**

| ObsMID | MMID |
|---|---|
| :obsM2 | :mM2 |

**Rainfall Observation**

| ObsID | ObsMID |
|---|---|
| :obs1 | :obsM1 |
| :obs2 | :obsM1 |
| :obs3 | :obsM1 |

**Temperature Observation**

| ObsID | ObsMID |
|---|---|
| :obs4 | :obsM2 |
| :obs5 | :obsM2 |
| :obs6 | :obsM2 |

**Rainfall Measurement**

| ObsID | MID |
|---|---|
| :obs1 | :m1 |
| :obs2 | :m2 |
| :obs3 | :m3 |

**Temperature Measurement**

| ObsID | MID |
|---|---|
| :obs4 | :m4 |
| :obs5 | :m5 |
| :obs6 | :m6 |

**Rainfall Instant**

| ObsID | Date Time |
|---|---|
| :obs1 | :time1 |
| :obs2 | :time2 |
| :obs3 | :time3 |

**Temperature Instant**

| ObsID | Date Time |
|---|---|
| :obs4 | :time4 |
| :obs5 | :time5 |
| :obs6 | :time6 |

**F-Measurement RDF-CT**

| MMID | Value | Unit |
|---|---|---|
| :mM1 | 20.0 | cm |
| :mM2 | 24.8 | °F |

**Instant RDF-CT**

| DateTime | Sampling Time |
|---|---|
| :time1 | ts1 |
| :time2 | ts2 |
| :time3 | ts3 |
| :time4 | ts4 |
| :time5 | ts5 |
| :time6 | ts6 |

(b) Factorized CT Based Parquet Tables

Figure 11: **CT based Tabular Representation of RDF Graphs**. Parquet tables are utilized to represent CT-based tabular representations of RDF graphs in Spark. (a) Each CT-based table stores a class template collected from the original RDF graph.(b) Factorized RDF graph is represented in compact CT based tables.

**Theorem 3.** *If G is an SSN RDF graph and G′ is a factorized RDF graph of G, and $T_1$ is the factorized tabular representation of G′, then $T_1$ is in third normal form with respect to the universal representation of G.*

*Proof.* Recall (Codd, 1972), a table is in third normal form if for every $X \rightarrow Y$

- X is a super key, or
- $Y - X$ is a prime attribute

Considering that the following functional dependencies hold in both the universal, and factorized tables:

- MMID → Value, Unit
- ObsMID → Type, Procedure, Property, MMID
- ObsID → SamplingTime, Timestamp, MID, ObsMID

It can be demonstrated that all the tables created after factorization are in 3NF. □

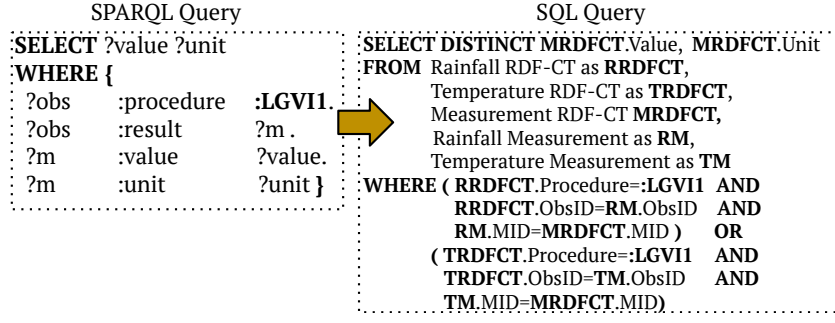**Theorem 4.** *The decomposition of the* Class Template (CT) *based tables representing sensor* data into the factorized CT *based tables is* loss-less join.

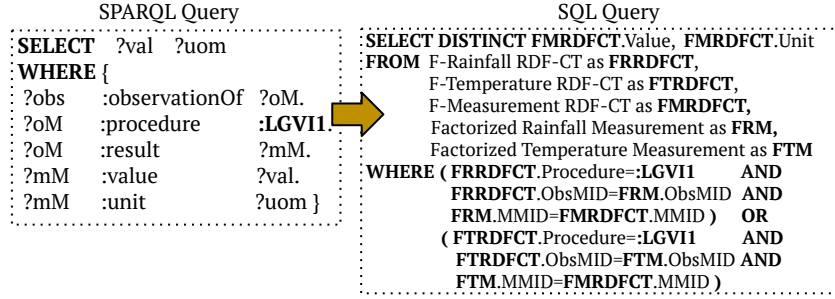*Proof.* Consider the following functional dependencies hold in CT and factorized CT tables:

- ObsMID → Procedure, Property
- MMID → Value, Unit
- ObsMID, MMID → ObsMID, MMID
- ObsID, ObsMID → ObsID, ObsMID
- ObsID, MID → ObsID, MID
- ObsID, SamplingTime → ObsID, SamplingTime
- SamplingTime → Timestamp

We can prove using the algorithm(Jeffrey, 1989) that the factorized CT based tables are a *loss-less join* decomposition of the CT based tables that includes all the attributes in the CT tables plus ObsMID and MMID. The attributes of the CT tables can be projected from G′, thus, satisfying the *loss-less join* condition. □

**Theorem 5.** *If G is an SSN RDF graph and G′ is a factorized RDF graph of G, and $T_2$ is the* Class Template (CT) *based tabular representation of G′,*

| SPARQL Query | SQL Query |
|---|---|

**SELECT** ?value ?unit
**WHERE {**
?obs     :procedure     **:LGVI1.**
?obs     :result        ?m .
?m       :value         ?value.
?m       :unit          ?unit **}**

**SELECT DISTINCT MRDFCT**.Value, **MRDFCT**.Unit
**FROM**  Rainfall RDF-CT as **RRDFCT**,
         Temperature RDF-CT as **TRDFCT**,
         Measurement RDF-CT **MRDFCT,**
         Rainfall Measurement as **RM,**
         Temperature Measurement as **TM**
**WHERE ( RRDFCT**.Procedure=**:LGVI1   AND**
         **RRDFCT**.ObsID=**RM**.ObsID    **AND**
         **RM**.MID=**MRDFCT**.MID **)**
       **( TRDFCT**.Procedure=**:LGVI1    AND**
         **TRDFCT**.ObsID=**TM**.ObsID    **AND**
         **TM**.MID=**MRDFCT**.MID**)**

(a) Query Class Template (CT) based Tables

| SPARQL Query | SQL Query |
|---|---|

**SELECT**   ?val   ?uom
**WHERE {**
?obs     :observationOf  ?oM.
?oM      :procedure      **:LGVI1.**
?oM      :result         ?mM.
?mM      :value          ?val.
?mM      :unit           ?uom **}**

**SELECT DISTINCT FMRDFCT**.Value, **FMRDFCT**.Unit
**FROM**  F-Rainfall RDF-CT as **FRRDFCT**,
         F-Temperature RDF-CT as **FTRDFCT**,
         F-Measurement RDF-CT as **FMRDFCT**,
         Factorized Rainfall Measurement as **FRM**,
         Factorized Temperature Measurement as **FTM**
**WHERE ( FRRDFCT**.Procedure=**:LGVI1      AND**
         **FRRDFCT**.ObsMID=**FRM**.ObsMID **AND**
         **FRM**.MMID=**FMRDFCT**.MMID **)    OR**
       **( FTRDFCT**.Procedure=**:LGVI1     AND**
         **FTRDFCT**.ObsMID=**FTM**.ObsMID **AND**
         **FTM**.MMID=**FMRDFCT**.MMID **)**

(b) Query Factorized CT based Tables

Figure 12: **Query Evaluation Over CT-based Tables**. Original and rewritten SPARQL queries and the corresponding SQL queries against CT-based tables are presented. (a) Query over CT-based tables of the original RDF graph. (b) The SQL query over CT-based tabular representation of the factorized RDF graph.

*then $T_2$ is in third normal form with respect to the* CT *based tabular representation of G.*

*Proof.* Recall (Codd, 1972), a table is in third normal form if for every $X \rightarrow Y$

- $X$ is a super key, or
- $Y - X$ is a prime attribute

Considering the following functional dependencies hold in CT based tables:

- ObsMID → Procedure, Property
- MMID → Value, Unit
- ObsMID, MMID → ObsMID, MMID
- ObsID, ObsMID → ObsID, ObsMID
- ObsID, MID → ObsID, MID
- ObsID, SamplingTime → ObsID, SamplingTime
- SamplingTime → Timestamp

It can be demonstrated that all the factorized tables are in 3NF. □

# 6 Experimental Study

We empirically study the effect of the proposed factorization techniques over RDF implementations accessible through RDF and Big Data engines. We evaluate the impact on the size of the factorized RDF graphs as well as on query execution time in different query engines. RDF-3X (Neumann and Weikum, 2010) is utilized to evaluate the influence of the proposed techniques over the RDF stores. Spark (Zaharia et al., 2016) is used to study the tabular representation of RDF graphs. In this work, we investigated the following research questions: **RQ1)** Are the proposed factorization techniques able to reduce the size of the semantically represented sensor data? **RQ2)** How is the factorization time affected by the size of the RDF graphs? **RQ3)** What is the impact of the queries against factorized RDF graphs over the query execution time? **RQ4)** Is the performance of queries against factorized RDF graphs affected by the size of the factorized RDF graphs or RDF implementation? The experimental configuration to evaluate the research questions mentioned above is as follows:
**Datasets:** Evaluation is conducted over two sensor datasets (Ali et al., 2015; Patni et al., 2010) described using the Semantic Sensor Network (SSN) Ontology. The RDF datasets describing weather observations are collected from around 20,000 weather stations in

**Algorithm 1:** The Incremental Factorization Algorithm

**Input:** An RDF graph $G(V_G, E_G, L_G)$, Previously factorized RDF Graph $G''(V_{G''}, E_{G''}, L_{G''})$, and entity mappings $\mu_{N''}$

**Output:** Factorized RDF Graph $G'(V_{G'}, E_{G'}, L_{G'})$, and entity mappings $\mu_N$

1   $\mu_N \longleftarrow \mu_{N''}, V_{G'} \longleftarrow V_{G''}, E_{G'} \longleftarrow E_{G''}, L_{G'} \longleftarrow L_{G''}$

2   **forall** $proc, ph, pp, val, uom \in V_G$ such that
    $SO = \{obs | (obs\ \texttt{rdf:type}\ ph) \in G,$
    $(obs\ \texttt{:procedure}\ proc) \in G,$
    $(obs\ \texttt{:property}\ pp) \in G, (obs\ \texttt{:result}\ m) \in$
    $G, (m\ \texttt{rdf:type}\ \texttt{:MeasureData}) \in G, (m\ \texttt{:unit}\ uom) \in$
    $G, (m\ \texttt{:value}\ val) \in G\}$, and
    $SM = \{m | (m\ \texttt{rdf:type}\ \texttt{:MeasureData}) \in$
    $G, (m\ \texttt{:unit}\ uom) \in G, (m\ \texttt{:value}\ val) \in G\}$ **do**

3    **if** $\exists mM, oM$ such that
    $(mM\ \texttt{rdf:type}\ \texttt{:MeasureData}) \in G'',$
    $(mM\ \texttt{:unit}\ uom) \in G'',$
    $(mM\ \texttt{:value}\ val) \in G'', (oM\ \texttt{rdf:type}\ ph) \in$
    $G'', (oM\ \texttt{:procedure}\ proc) \in$
    $G'', (oM\ \texttt{:property}\ pp) \in G'',$ and
    $(oM\ \texttt{:result}\ mM) \in G''$ **then**

4      **foreach**
      $(s\ \texttt{rdf:type}\ o) \in E_G \wedge s, o \in V_G \wedge \texttt{rdf:type} \in L_G$
      such that $s \in SM \cup SO$ **do**

5        **if** $s \in SM$ **then**

6          $\mu_N \leftarrow \mu_N \cup \{(s, mM)\}$

7        **else**

8          $\mu_N \leftarrow \mu_N \cup \{(s, oM)\}, E_{G'} \leftarrow$
         $E_{G'} \cup (s\ \texttt{:observationOf}\ \mu_N(s))\}$

9          $V_{G'} \leftarrow V_{G'} \cup \{s, o\},$
         $L_{G'} \leftarrow L_{G'} \cup \{\texttt{:observationOf}\}$

10    **else**

11      $mM \leftarrow SurrogateMeasurement(), oM \leftarrow$
       $SurrogateObservation()$

12      **foreach** $m \in SM$ **do**

13        $\mu_N \leftarrow \mu_N \cup \{(m, mM)\}$

14      **foreach** $obs \in SO$ **do**

15        $\mu_N \leftarrow \mu_N \cup \{(obs, oM)\}$

16    **foreach** $(s\ p\ o) \in E_G \wedge s, o \in V_G \wedge p \in L_G$ **do**

17      **if** $s \in SM \cup SO$ **then**

18        **if** $p == \texttt{:result}$ **then**

19          $E_{G'} \leftarrow E_{G'} \cup$
         $\{(s\ p\ o), (\mu_N(s)\ p\ \mu_N(o)), (s\ \texttt{:observationOf}\ \mu_N(s))\}$

20          $V_{G'} \leftarrow V_{G'} \cup \{s, o, \mu_N(s), \mu_N(o)\},$
         $L_{G'} \leftarrow L_{G'} \cup \{p, \texttt{:observationOf}\}$

21        **else if** $p == \texttt{rdf:type}\ \&\&\ (o ==$
       $\texttt{:Observation} || o == \texttt{:MeasureData})$
       **then**

22          $E_{G'} \leftarrow E_{G'} \cup \{(\mu_N(s)\ p\ o)\},$
         $V_{G'} \leftarrow V_{G'} \cup \{\mu_N(s), o\}, L_{G'} \leftarrow L_{G'} \cup \{p\}$

23        **else if** $p == \texttt{:procedure} || p ==$
       $\texttt{:property} || p == \texttt{:value} || p == \texttt{:unit}$
       **then**

24          $E_{G'} \leftarrow E_{G'} \cup \{(\mu_N(s)\ p\ o)\}, V_{G'} \leftarrow$
         $V_{G'} \cup \{\mu_N(s), o\}, L_{G'} \leftarrow L_{G'} \cup \{p\}$

25        **else**

26          $E_{G'} \leftarrow E_{G'} \cup \{s\ p\ o)\}, V_{G'} \leftarrow V_{G'} \cup \{s, o\},$
         $L_{G'} \leftarrow L_{G'} \cup \{p\}$

27   **return** $G'(V_{G'}, E_{G'}, L_{G'}), \mu_N$

---

**Algorithm 2:** The Query Rewriting Algorithm

**Input:** Set $ST$ of triple patterns in a BGP of $Q$ and set $SR$ of query rewriting rules

**Output:** $ST_{new}$ the rewriting of $ST$ under $SR$

1   $ST_{new} \longleftarrow \emptyset$

2   **foreach** $t \in ST$ **do**
- Select $r \in SR$ such that $t$ matches the head of $r$ and instantiate the body of $r$
- Let $SQ_t$ be the matched body of $r$ and *variableSubstitutions* be the set of mappings between variables in $t$ into $SQ_t$, add $(t, SQ_t, variableSubstitutions)$ to $ST_{new}$

3   **return** $ST_{new}$

---

the United States[5]. Realtime smart city datasets are collected from the city of Aarhus, Denmark. The smart city datasets encompasses the traffic, pollution, and parking observations [6]. Table 2 describes the main characteristics of these RDF datasets.

**Queries:** The SRBench-Version 0.9 queries[7] are used as baseline in our experimental testbed. Because RDF-3X does not evaluate queries with the OPTIONAL operator, query 2 is modified to include only one BGP. Also, the STREAM clause, ASK queries, aggregate modifiers like AVG, GROUP BY, and HAVING are not supported. So, only SELECT queries without aggregate modifiers are part of our testbed. Queries range from simple queries with one triple pattern to complex queries having up to 14 triple patterns with UNION and FILTER clauses [8].

**Metrics:** We report on the following metrics: **a) Number of Triples (NT)** in the semantic sensor data collection. **b) Percentage Savings (%age NT Savings)** in the number of RDF triples after factorization; higher the better. **c) Average Number of Triples per Observation (avg. NT per Obs.)** represents the average number of RDF triples describing an observation; lower the better. **d) Factorization Time (FT)** is the elapsed time between the request of factorization and the generation of the factorized RDF graph. **e) RDF3x Loading Time (LT)** is the time required to load RDF data to RDF3x store. **FT** and **LT** are computed as the *real time* of the *time* command of the Linux operating system. **f) Query Execution Time**

Table 2: **Datasets**: Description of the semantic sensor datasets; weather and smart city datasets; collected from the United States and Aarhus, Denmark, respectively.

| | Weather Dataset | | | Smart City Dataset | | |
|---|---|---|---|---|---|---|
| ID | Climate Event | #Triples | # Obs | ID | #Triples | # Obs |
| D1 | Blizzard | 38,054,493 | 4,092,492 | C1 | 47,487,800 | 4,748,884 |
| D2 | Hurricane Charley | 108,644,568 | 11,648,607 | C2 | 47,051,850 | 4,705,267 |
| D3 | Hurricane Katrina | 179,128,407 | 19,233,458 | C3 | 56,816,196 | 5,681,712 |

Table 3: **Effectiveness of the Semantic Sensor Data Factorization**. Number of triples (**NT**) before and after factorization along with %age NT savings.

| Dataset ID | Number of Triples(NT) | | %age NT Savings | Avg. NT per Obs. | |
|---|---|---|---|---|---|
| | Original | Factorized | | Original | Factorized |
| **D1** | 38,054,493 | 17,800,156 | 53.22 | 9.29 | 4.34 |
| **D1D2** | 146,699,061 | 63,993,774 | 56.38 | 9.32 | 4.06 |
| **D1D2D3** | 325,827,468 | 136,979,696 | **57.96** | 9.31 | **3.92** |
| **C1** | 47,487,800 | 23,937,396 | 49.59 | 9.99 | 5.04 |
| **C1C2** | 94,539,650 | 47,621,691 | 49.63 | 9.99 | 5.04 |
| **C1C2C3** | 151,355,846 | 76,223,192 | **49.64** | 9.99 | **5.04** |

(**ET**) is the elapsed time between the submission of the query to the engine and the complete output of the answer, and is measured as the *real time* produced by the *time* command of the Linux operation system.

**Implementation:** Three series of experiments were conducted over the gradually integrating sensor datasets in Table 2, i.e., D1, D1D2, and D1D2D3. **i)** Algorithm 1 is executed over the original RDF datasets to generate the factorized RDF representations. Moreover, original and factorized RDF datasets are loaded in RDF3X store. **ii)** SPARQL queries are executed using RDF3X engine over original and factorized RDF datasets. The experiments are executed on a Linux Debian 8 machine with a CPU Intel I7 980X 3.3GHz and 32GB RAM 1333MHz DDR3. Queries are run on both cold and warm cache.[9] to assess the query performance when data is cached. To run on warm cache, we executed the same query five times by dropping the cache just before running the first iteration of the query; thus, data temporally stored in cache during the execution of iteration $i$ can be used in iteration $i+1$. **iii)** In the third series of experiments, SQL queries were run on cold and warm cache using *Apache Spark*[10] over the universal, factorized, original and factorized CT-based tabular representations. These tabular representations are stored using Parquet format in *HDFS*[11]. The experiments were conducted on a spark cluster of one master and three worker nodes.The experiments are performed

on a machine with Intel(R) Xeon(R) Platinum 8160 CPU 2.10GHz and 23 RAM slots, where each RAM slot is DDR4 type, 32GB RAM size, and 2666MHz speed. The source code of the factorization approach is available on github[12].

**Efficiency and Effectiveness of Factorized RDF.** For evaluating the efficiency and effectiveness of the proposed factorization techniques and to answer the research questions **RQ1** and **RQ2**, we execute algorithm 1 by gradually integrating the datasets in Table 2, i.e., **D1**, **D1D2**, and **D1D2D3**. Effectiveness is reported based on the reduction of RDF triples (**NT**), while efficiency is measured in terms of factorization time (**FT**) and RDF3X loading time (**LT**). Table 3 reports on the number of RDF triples (**NT**) in datasets **D1**, **D1D2**, and **D1D2D3** before and after the factorization, as well as in datasets **C1**, **C1C2**, and **C1C2C3**. The results demonstrate that the proposed factorization techniques are capable of reducing the RDF triples by at least **53.22%** in the datasets of weather observations, and **49.59%** in smart city dataset. Moreover, the results report that the factorized representation of sensor observations requires in average a small number of RDF triples, e.g., five RDF triples instead of ten in the weather dataset, while preserving all the information within the original RDF graph. These results allows us to positively answer research question **RQ1**, i.e., factorized RDF graphs effectively reduce the size of RDF graphs. We also measure factorization time and factorized RDF loading time to RDF3X, and compare to the time required

---

[9]To run cold cache, we clear the cache before running each query by performing the command `sh -c "sync ; echo 3 > /proc/sys/vm/drop_caches"`

[10]`http://spark.apache.org/`

[11]https://hadoop.apache.org/

[12]`https://github.com/SDM-TIB/SemanticSensorDataFactorization`

Table 4: **Efficiency of the Semantic Sensor Data Factorization**. Time that elapses during factorization (**FT**) as well as the RDF3X Loading Time (**LT**).

| Dataset ID | Factorization Time FT(s) | RDF3X LT(s) | |
|---|---|---|---|
| | | Original | Factorized |
| **D1** | 417.229 | 460.511 | 252.976 |
| **D1D2** | 1,260.495 | 1,887.626 | 970.150 |
| **D1D2D3** | 2,147.239 | 3,822.723 | 1,982.697 |



Figure 13: **Query Execution Time ET (ms Log-scale) over RDF3x**. Original SPARQL queries $Q$ and rewritten SPARQL queries $Q'$ are evaluated on **cold** cache against original and factorized RDF graphs, respectively. Rewritten queries reduce execution time on factorized RDF graphs by up one order of magnitude.

by RDF3X to upload the original RDF graphs, in Table 4. Algorithm 1 as well as factorized RDF loading to RDF3X requires less than **50%** of the time consumed by RDF3X during original RDF data loading. Thus, with these results research question **RQ2** can be also positively answered.

**Impact of Factorized RDF over Query Processing.** We analyze the efficiency of the proposed representations by running the queries generated using Algorithm 2. First, the impact of our approach on query execution is studied over centralized RDF engines; to evaluate the benefits of caching previous results, queries are executed on cold and warm caches. The advantage of running these queries on cold and warm caches on RDF3X are analyzed over the gradually increasing original and factorized RDF datasets. The original queries Q are compared to the reformulated queries Q′. Original queries (Q) are executed against the original datasets, while plans for reformulated queries (Q′) are run against gradually increasing factorized datasets. Figure 13 reports on the query execution time (milliseconds. log-scale) with cold cache, while Figure 14 depicts the observed execution time when queries are run on warm cache; the minimum value is reported in all the queries. In all cases, reformulated queries over factorized RDF graphs exhibit better performance whenever they are run on cold and warm caches. This observation supports the statement that because observation and measurement multiplicity is reduced to one in factorized RDF graphs, factorized queries produce small intermediate results which can be maintained in resident memory and re-used in further executions. Thus, the performance of factorized queries is considerable better with warm cache,

overcoming other executions by up to three orders of magnitude, e.g., Q2 and Q6. Results also suggest that performance of reformulated queries is not affected by the RDF graph size, e.g., large RDF graphs like D1D2D3 with 325,827,468 RDF triples.

We further analyse the effect of factorization when query processing is conducted over the relational representations of sensor data, i.e., universal and factorized tables, and the CT based tabular implementation of original and factorized RDF data. The performance of queries over Parquet tables depends on the number of attributes included in the query, as well as on the ratio between the attributes in the query and the attributes in the tables [13]. In queries against the universal table, the ratio between the number of attributes varies from **0.09** to **0.45**. While the ratio in factorized queries is in the range from **0.46** to **0.75**, and in original and factorized CTs is **0.25** and **0.78**. So, based on this statement, queries over the universal table should be faster than queries over the factorized tables and CT based tables. However, as observed in Figures 15 and 16, reformulated queries over factorized CT tables speed up execution time to almost two orders of magnitude, except Q11 where factorized tables are performing better. Factorized CTs reduce the size of tables by creating them around each molecule template and factorization further removes data redundancies. Actually, in queries Q4 and Q5, execution over the universal table times out after 100 minutes. These results indicate that the rewritten queries speed up query processing over big data engines.

**Discussion.** The presented experimental results confirm that the factorization techniques are able to re-
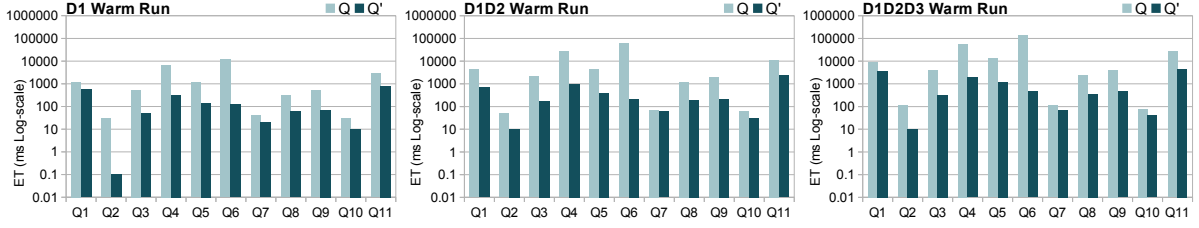
---

[13]https://parquet.apache.org/

Figure 14: **Query Execution Time ET (ms Log-scale) over RDF3x**. SPARQL queries *Q* and rewritten queries *Q'* are evaluated on **warm** cache against original and factorized RDF graphs, respectively. To warm cache up, memory is flushed. The rewritten queries reduce query execution time by up two order of magnitude.
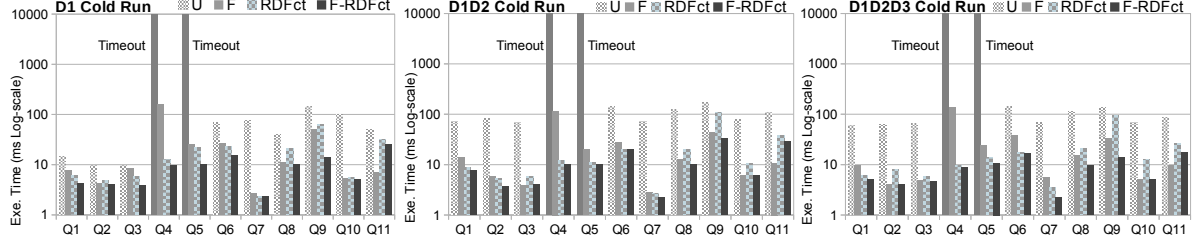


Figure 15: **Query Execution Time ET (ms Log-scale) over Relations**. Query evaluation over tabular based representations in **cold** cache. Executions are timed out after 100 minutes. SQL version of the rewritten SPARQL queries over the factorized (*F*) and factorized CT tables (F-RDFct) reduce execution time.
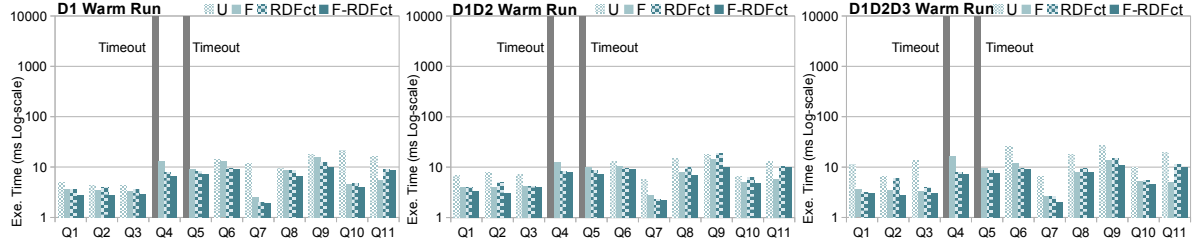


Figure 16: **Query Execution Time ET (ms Log-scale) over Relations**. Query evaluation over tabular representations in **warm** cache. Execution timeout is 100 minutes. SQL queries execution time over the factorized (*F*) and factorized CT tables (F-RDFct) is less than the universal (U) and original CT tables (RDFct).

duce duplicated measurements in observational data without any information lost. Furthermore, since graphs can be factorized incrementally, savings are observed whenever new incoming tuples are related to measures previously collected. The benefits of the approach are reported in the reduction of the number of RDF triples of the factorized graphs, as well as in the execution time of queries rewritten over these factorized graphs. These savings are even more significant when the query engine provides efficient caching techniques to maintain in cache intermediate results of previously evaluated queries. Lastly, in the case of relational representation of factorized data in big data infrastructures, space savings are significant, enabling an efficient query execution over factorized tables.

# 7 Conclusions and Future Work

This article presents compact RDF representations for semantic sensor data to reduce data redundancy, while information is preserved and query execution performance is enhanced. Moreover, the effectiveness of the proposed approach was studied over several query engines. Furthermore, tabular representations for a loss-less large-scale storage of factorized semantic sensor data are presented. A factorization algorithm transforms original observations and measurements to a compact representation where data redundancy is reduced. Additionally, query rewriting rules and a query re-writing algorithm are presented. The query rewriting algorithm exploits the rewriting rules to rewrite SPARQL queries against factorized RDF graphs, and speeds up query execution time. The factorized observations and measurements are also exploited to produce tabular representations for factor-

ized RDF graphs utilizing Parquet tables. We empirically evaluate the effectiveness of the proposed factorization techniques and results confirm that exploiting semantics encoded in semantic sensor data allow for reducing redundancy by up to 57.96%, while the time taken by the process of factorizing RDF data is less than 50% of loading time for the original RDF data in state-of-the-art RDF stores. Further, the loading time for factorized RDF data is reduced by more than 45% of the loading time of original RDF data in native RDF stores. Also, we evaluated the impact of proposed compact representations over the diverse implementations available for RDF data, i.e., native RDF implementations and non-native large-scale tabular based implementations. Thus, *CSSD* broadens the portfolio of tools that enable to semantically enrich sensor data. As the main limitation, *CSSD* can only be applied to data coming from one single device. In the future, we plan to devise data integration techniques able to merge RDF molecules generated from the factorization of heterogeneous data collected either from sensors or static data sources of observational data. We will apply these techniques to the energy domain to facilitate the integration and analysis of data collected from diverse energy providers.

## Acknowledgments

## REFERENCES

Ali, M. I., Gao, F., and Mileo, A. (2015). Citybench: a configurable benchmark to evaluate rsp engines using smart city datasets. In *International Semantic Web Conference*, pages 374–389. Springer.

Álvarez-García, S., Brisaboa, N. R., Fernández, J. D., and Martínez-Prieto, M. A. (2011). Compressed k2-triples for full-in-memory rdf engines. *arXiv preprint arXiv:1105.4004*.

Arenas, M., Gutierrez, C., and Pérez, J. (2009). Foundations of rdf databases. In *Reasoning Web. Semantic Technologies for Information Systems*, pages 158–204. Springer.

Bakibayev, N., Kociský, T., Olteanu, D., and Zavodny, J. (2013). Aggregation and ordering in factorised databases. *PVLDB*, 6(14):1990–2001.

Bakibayev, N., Olteanu, D., and Zavodny, J. (2012). FDB: A query engine for factorised relational databases. *PVLDB*, 5(11):1232–1243.

Bok, K., Han, J., Lim, J., and Yoo, J. (2019). Provenance compression scheme based on graph patterns for large rdf documents. *The Journal of Supercomputing*, pages 1–23.

Brayton, R. K. (1987). Factoring logic functions. *IBM Journal of research and development*, 31(2):187–198.

Brisaboa, N. R., Ladra, S., and Navarro, G. (2009). k2-trees for compact web graph representation. In *International Symposium on String Processing and Information Retrieval*, pages 18–30. Springer.

Codd, E. F. (1972). Further normalization of the data base relational model. *Data base systems*, pages 33–64.

Compton, M., Barnaghi, P., Bermudez, L., GarcíA-Castro, R., Corcho, O., Cox, S., Graybeal, J., Hauswirth, M., Henson, C., Herzog, A., et al. (2012). The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25–32.

Copeland, G. P. and Khoshafian, S. N. (1985). A decomposition storage model. In *Acm Sigmod Record*, volume 14, pages 268–279. ACM.

Du, J.-H., Wang, H.-F., Ni, Y., and Yu, Y. (2012). Hadooprdf: A scalable semantic data analytical engine. In *International Conference on Intelligent Computing*, pages 633–641. Springer.

Endris, K. M., Galkin, M., Lytra, I., Mami, M. N., Vidal, M.-E., and Auer, S. (2017). Mulder: querying the linked data web by bridging rdf molecule templates. In *International Conference on Database and Expert Systems Applications*, pages 3–18. Springer.

Fernández, J. D., Llaves, A., and Corcho, Ó. (2014). Efficient RDF interchange (ERI) format for RDF data streams. In *The Semantic Web - ISWC 2014*, pages 244–259.

Fernández, J. D., Martínez-Prieto, M. A., Gutiérrez, C., Polleres, A., and Arias, M. (2013). Binary RDF representation for publication and exchange (HDT). *J. Web Sem.*, 19:22–41.

Gaur, A., Scotney, B., Parr, G., and McClean, S. (2015). Smart city architecture and its applications based on iot. *Procedia computer science*, 52:1089–1094.

Idreos, S., Groffen, F., Nes, N., Manegold, S., Mullender, S., and Kersten, M. (2012). Monetdb: Two decades of research in column-oriented database. *IEEE Data Engineering Bulletin*.

Jabbar, S., Ullah, F., Khalid, S., Khan, M., and Han, K. (2017). Semantic interoperability in heterogeneous iot infrastructure for healthcare. *Wireless Communications and Mobile Computing*.

Jeffrey, D. U. (1989). Principles of database and knowledge-base systems.

Joshi, A. K., Hitzler, P., and Dong, G. (2013). Logical linked data compression. In *10th Extended Semantic Web Conf. ESWC*, pages 170–184.

Karim, F., Mami, M. N., Vidal, M.-E., and Auer, S. (2017). Large-scale storage and query processing for semantic sensor data. In *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics*, page 8. ACM.

Khadilkar, V., Kantarcioglu, M., Thuraisingham, B., and Castagna, P. (2012). Jena-hbase: A distributed, scalable and efficient rdf triple store. In *Proceedings of the 11th International Semantic Web Conference Posters & Demonstrations Track, ISWC-PD*, volume 12, pages 85–88. Citeseer.

MacNicol, R. and French, B. (2004). Sybase iq multiplex-designed for analytics. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1227–1230. VLDB Endowment.

Mami, M. N., Scerri, S., Auer, S., and Vidal, M.-E. (2016). Towards semantification of big data technology. In *International Conference on Big Data Analytics and Knowledge Discovery*, pages 376–390. Springer.

Meier, M. (2008). Towards rule-based minimization of rdf graphs under constraints. In *International Conference on Web Reasoning and Rule Systems*, pages 89–103. Springer.

Neumann, T. and Weikum, G. (2010). The rdf-3x engine for scalable management of rdf data. *The VLDB Journal The International Journal on Very Large Data Bases*, 19(1):91–113.

Nie, Z., Du, F., Chen, Y., Du, X., and Xu, L. (2012). Efficient sparql query processing in mapreduce through data partitioning and indexing. In *Asia-Pacific Web Conference*, pages 628–635. Springer.

Pan, J. Z., Gómez-Pérez, J. M., Ren, Y., Wu, H., Wang, H., and Zhu, M. (2014). Graph pattern based RDF data compression. In *4th Joint Int. Conf. on Semantic Technology (JIST)*.

Papailiou, N., Konstantinou, I., Tsoumakos, D., Karras, P., and Koziris, N. (2013). H 2 rdf+: High-performance distributed joins over large-scale rdf graphs. In *2013 IEEE International Conference on Big Data*, pages 255–263. IEEE.

Patni, H., Henson, C., and Sheth, A. (2010). Linked sensor data. In *Collaborative Technologies and Systems (CTS), 2010 International Symposium on*, pages 362–370. IEEE.

Pichler, R., Polleres, A., Skritek, S., and Woltran, S. (2010). Redundancy elimination on rdf graphs in the presence of rules, constraints, and queries. In *International Conference on Web Reasoning and Rule Systems*, pages 133–148. Springer.

Punnoose, R., Crainiceanu, A., and Rapp, D. (2012). Rya: a scalable rdf triple store for the clouds. In *Proceedings of the 1st International Workshop on Cloud Intelligence*, page 4. ACM.

Schätzle, A., Przyjaciel-Zablocki, M., Dorner, C., Hornung, T., and Lausen, G. (2012). Cascading map-side joins over hbase for scalable join processing. In *SSWS+ HPCSW@ ISWC*, pages 59–74.

Schätzle, A., Przyjaciel-Zablocki, M., Hornung, T., and Lausen, G. (2013). Pigsparql: A sparql query processing baseline for big data. In *International Semantic Web Conference (Posters & Demos)*, volume 1035, pages 241–244.

Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., Lau, E., Lin, A., Madden, S., O'Neil, E., et al. (2005). C-store: a column-oriented dbms. In *Proceedings of Very large data bases*, pages 553–564. VLDB Endowment.

Ullman, J. D. (1984). *Principles of database systems*. Galgotia publications.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. (2016). Apache spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65.

Zukowski, M., Heman, S., Nes, N., and Boncz, P. A. (2006). Super-scalar ram-cpu cache compression. In *Icde*, volume 6, page 59.