

Compacting Frequent Star Patterns in RDF Graphs

FARAH KARIM

Leibniz University of Hannover, Germany
 Mirpur University of Science and Technology (MUST), Mirpur-10250 (AJK), Pakistan
 Karim@l3s.de

MARIA-ESTHER VIDAL

TIB Leibniz Information Centre for Science and Technology, Hannover
 Leibniz University of Hannover, Germany
 Maria.Vidal@tib.eu

SÖREN AUER

TIB Leibniz Information Centre for Science and Technology, Hannover
 Leibniz University of Hannover, Germany
 Auer@tib.eu

March 12, 2020

Abstract

Knowledge graphs have become a popular formalism for representing entities and their properties using a graph data model, e.g., the Resource Description Framework (RDF). An RDF graph comprises entities of the same type connected to objects or other entities using labeled edges annotated with properties. RDF graphs usually contain entities that share the same objects in a certain group of properties, i.e., they match star patterns composed of these properties and objects. In case the number of these entities or properties in these star patterns is large, the size of the RDF graph and query processing are negatively impacted; we refer these star patterns as *frequent star patterns*. We address the problem of identifying *frequent star patterns* in RDF graphs and devise the concept of *factorized RDF graphs*, which denote compact representations of RDF graphs where the number of frequent star patterns is minimized. We also

develop computational methods to identify frequent star patterns and generate a *factorized RDF graph*, where *compact RDF molecules* replace frequent star patterns. A compact RDF molecule of a frequent star pattern denotes an RDF subgraph that instantiates the corresponding star pattern. Instead of having all the entities matching the original frequent star pattern, a *surrogate* entity is added and related to the properties of the frequent star pattern; it is linked to the entities that originally match the frequent star pattern. Since the edges between the entities and the objects in the frequent star pattern are replaced by edges between these entities and the surrogate entity of the compact RDF molecule, the size of the RDF graph is reduced. We evaluate the performance of our factorization techniques on several RDF graph benchmarks and compare with a baseline built on top of *gSpan*, a state-of-the-art algorithm to detect frequent patterns. The outcomes evidence the efficiency of proposed approach and show that our tech-

niques are able to reduce execution time of the baseline approach in at least three orders of magnitude. Additionally, RDF graph size can be reduced by up to 66.56% while data represented in the original RDF graph is preserved.

Keywords: Semantic Web, RDF Compaction, Linked Data, Knowledge Graph.

1 Introduction

Knowledge graphs have gained momentum as flexible and expressive structures for representing not only data and knowledge but also actionable insights [28]; they provide the basis for effective and intelligent applications. Currently, knowledge graphs are utilized in diverse domains e.g., DBpedia [19], Google Knowledge Graph [26], and KnowLife [12]. The Resource Description Framework (RDF) [18] has been adopted as a formalism to represent knowledge graphs; in fact, in the Linked Open Data cloud [6], there are in 2019 more than 1,200 RDF knowledge graphs available¹. RDF models knowledge in the form of graphs where nodes represent entities; connections between entity nodes are representing RDF triples composed of subject, property, and object. The subjects and objects are represented by nodes, and an edge represents a property that relates a subject with an object. Diverse applications have been developed on top of knowledge graphs [5, 15, 28]. However, the adoption of knowledge graphs as *de facto data* structure of real-world applications demands efficient representations and scalable techniques for creating, managing, and answering queries over knowledge graphs. Thus, efficient graph representations of real-world scenarios are still demanded to enhance and facilitate the development of applications over knowledge graphs.

In real-world applications, a group of entities can share the same values in a set of features. For example, several sensor observations can sense the same temperature, in a given timestamp and city. This situation can be represented in an RDF graph with four triples per sensor observation o_i , i.e., $(o_i \text{ temperature } t)$, $(o_i \text{ unit } u)$, $(o_i \text{ timestamp } ts)$,

and $(o_i \text{ gps_coordinates } gc)$. All the resources representing these sensor observations match the variable $?o$ in the star pattern (SGP) composed by the conjunction of the following triple patterns $(?o \text{ temperature } t)$ $(?o \text{ unit } u)$, $(?o \text{ timestamp } ts)$, and $(?o \text{ gps_coordinates } gc)$ [24]. In case the star patterns are instantiated with many entities, a large number of RDF triples will have the same properties and objects and the corresponding star pattern will be repeatedly instantiated; we name these star patterns *frequent star patterns*. Although RDF triples that instantiate a frequent star pattern correctly model the real world, the size of the knowledge graph as well as the efficiency of the tasks of management and processing, can be negatively affected whenever a large number of triples of frequent star patterns populate the knowledge graph. Since frequent star patterns are very common in real world knowledge graphs, techniques are required to enable both the efficient representation of the knowledge encoded in these star patterns, as well as the processing and traversal of the represented knowledge.

The Database and Semantic Web communities have addressed the problem of representing relational and graph data models; they have proposed a variety of representation methods and data structures that take into account the main features of a relational or graph model with the aim of speeding up relation and graph based analytics [1, 2, 3, 14, 16, 17, 20, 23, 32]. Compression techniques [1, 32] over the column-oriented databases [7, 27], use the decomposition storage model [10] to maintain data, where each attribute value and a surrogate key, from the conceptual schema, are stored in a binary relation. However, a relation stored using the decomposition storage model cannot easily exploit compression unless surrogate keys are repeated [10]. Further, the decomposition model stores two copies of a binary relation, also the surrogate keys are required to be stored repeatedly for each attribute causing an increase in the storage space requirements. In the context of RDF graph, the scientific community has also actively contributed; approaches like [3, 14, 21, 31] generate compact binary representations for RDF knowledge graphs. RDF binary compression techniques do not take into account the semantics encoded in knowl-

¹<https://lod-cloud.net/>

edge graphs; they require customized engines to perform query processing. Moreover, there have been defined compression approaches for RDF graphs able to exploit semantics encoded in RDF triples. Approaches [20, 23] are application dependent and require a user to input the compression rules and constraints. Alternatively, compression approaches tailored for ontology properties [17] have shown to be effective, but they require prior knowledge of classes and properties involved in repeated graph patterns to generate compact representations. Lastly, techniques proposed by Joshi et al. [16] require decompression to access and process the original data, as well as extra processing over the data. Albeit effective in reducing the storage space, existing compression methods add overhead to the process of data management, and particularly, query execution time can be negatively impacted. gSpan [30] and GRAMI [11] are state-of-the-art algorithms that aim to identify frequent patterns. However, only patterns with constants are considered and they are neither able to identify star patterns nor decide *frequentness*. We have built an exhaustive algorithm that resorts to the gSpan enumeration of frequent patterns to identify the frequent star patterns in an RDF knowledge graph; this approach corresponds to the baseline of our empirical evaluation.

Our Research Goal: We address the problem of *identifying frequent star patterns* in RDF knowledge graphs, where certain properties and their corresponding objects are repeatedly shared by several entities of a type causing unnecessary growth of the knowledge graphs. Our research goal is to minimize the number of *frequent star patterns* in RDF knowledge graphs to generate compact representations without losing any information. We investigate the following research questions:

- What are the criteria that characterize frequent star patterns?
- Do compact graph representations impact on the size of knowledge graphs?

Approach: We devise the concept of *factorized RDF graphs*, which corresponds to a compact graph with a minimized number of frequent star patterns. Further,

we develop computational methods to detect frequent star patterns in RDF graphs and to generate a *factorized RDF graph*. These methods are able to identify entities and properties in frequent star patterns in RDF graphs, and generate factorized RDF graphs by representing frequent star patterns with compact RDF molecules. A compact RDF molecule of a frequent star pattern is an RDF subgraph that instantiates the star pattern; a surrogate entity stands for the entities that satisfy the corresponding frequent star pattern. The surrogate entity is linked to the properties and the corresponding objects in the frequent star pattern (see Figure 4c). The entities, initially matching the frequent star pattern, are also linked to the surrogate entity of the compact RDF molecule. Compact RDF molecules significantly reduce the size of the RDF graph by replacing labeled edges and entities connected the objects in the frequent star pattern, with edges linking the entities to the surrogate entity of a compact RDF molecule. We study the effectiveness of our factorization techniques over the *LinkedSensorData* benchmark [22]; it describes more than 34,000,000 weather observations collected by around 20,000 weather stations in the United States since 2002. Experiments are conducted against three *LinkedSensorData* RDF graphs by gradually increasing the graph size. The observed results evidence that frequent star patterns characterize the best set of properties relating several entities of a class to the same objects in an RDF graph. Moreover, our techniques reduce RDF graphs size by up to 66.56% using properties and classes recommended by the frequent star patterns detection approach.

Contributions: we devise computational methods for factorizing RDF graphs. The specific contributions are as follows: **i)** Criteria for detecting frequent star patterns; **ii)** Factorization techniques compacting frequent star patterns in RDF graphs. We have presented two algorithms: An exhaustive approach (named E.FSP) searches the space of frequent patterns produced by an algorithm like gSpan, to identify frequent star patterns. Further, G.FSP implements a Greedy meta-heuristics that is able to traverse the space of star patterns and identify the ones that are frequent. Star patterns are traversed in iterations, starting with the star patterns with the

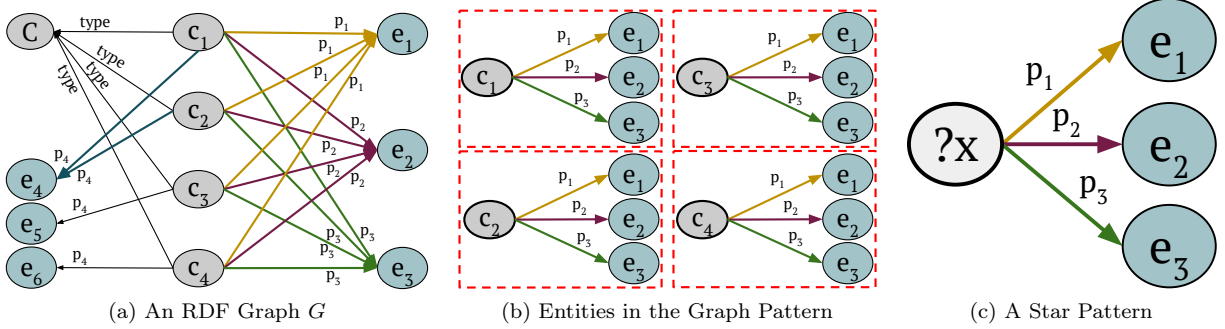


Figure 1: **Motivating Example.** Frequent star pattern. (a) RDF graph with classes, entities, and properties; (b) Entities c_1, c_2, c_3 , and c_4 are related to e_1, e_2 , and e_3 with properties p_1, p_2 , and p_3 , respectively; (c) A star pattern with subject variable $?x$, respectively, relates e_1, e_2 , and e_3 with properties p_1, p_2 , and p_3 .

largest number of properties. The criteria of frequent star patterns correspond the stop criteria of the algorithm. **iii)** An empirical study of both the frequent star patterns detection and factorization techniques using existing benchmarks. Experimental results show that both E.FSP and G.FSP identify frequent star patterns. Moreover, G.FSP overcomes E.FSP by reducing execution time in at least three orders of magnitude. More importantly, the experiments indicate that factorizing frequent star patterns by using surrogate keys enable for the creation of compact RDF graphs that reduce size while preserving the information in the original RDF graph.

The article is structured as follows: We motivate our research in Section 2, and present an analysis of the state of the art in Section 3. Our approach is defined in Section 4, while Section 5 reports on the results of the experimental study. Finally, we conclude with an outlook on future work in Section 6.

2 Motivating Example

We motivate the problem addressed by this work with an RDF graph where entities of the same type – or resources – match the same star pattern. In an RDF graph, matching the same star pattern means that the properties and objects are the same, whereas the

entities are different. When the number of entities matching a star pattern is very high, the size of the RDF graph increases and the query processing over the RDF graph is affected negatively. A star pattern with a high number of matching entities is a frequent star pattern. Figure 1a depicts an RDF graph composed by a class C , the entities $c_1, c_2, c_3, c_4, e_1, e_2, e_3, e_4, e_5$, and e_6 , and the properties p_1, p_2, p_3 , and p_4 . A directed edge ($s p o$) in the RDF graph stands for an RDF triple where p is a label that represents an RDF predicate, while s and o are subject and object nodes, respectively. Edges labeled with the predicate *type*², indicate that c_1, c_2, c_3 and c_4 are of the same type, i.e., the class C . The directed edge ($c_1 p_1 e_1$) expresses that the entity c_1 is related to object e_1 with the property p_1 . Similarly, entities c_2, c_3 , and c_4 are related to object e_1 with the property p_1 , i.e., the indegree of e_1 is four. Similarly, entities c_1, c_2, c_3 and c_4 are related to e_2 and e_3 with the properties p_2 and p_3 , respectively. Note that entities c_1, c_2, c_3 , and c_4 are associated with the same objects, i.e., e_1, e_2 and e_3 through the edges annotated with same properties p_1, p_2 , and p_3 . Albeit sound, these redundant labeled edges generate frequent star patterns because entities of the same type are described using the same properties and objects. Figure 1b illustrates the RDF subgraphs that map to the same

²property *type* refers to *rdf:type*

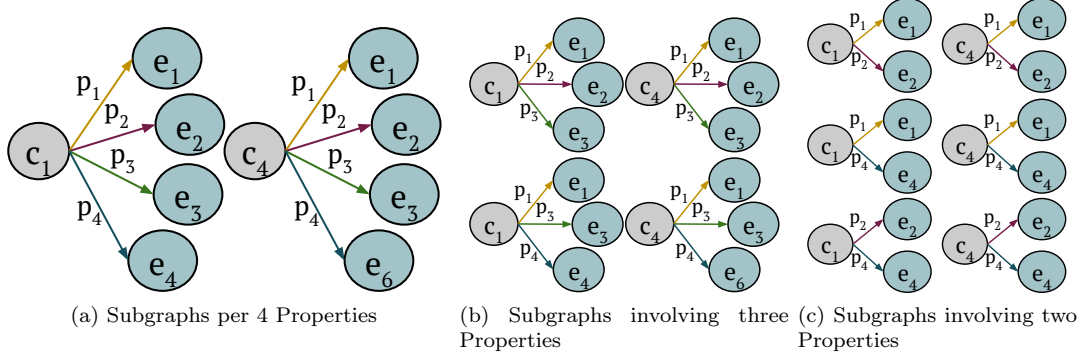


Figure 2: **Graph Patterns Identified by gSpan.** Subgraphs, involving entities c_1 and c_4 , extracted by gSpan from the RDF graph in Figure 1a. (a) Subgraphs per set $\{p_1, p_2, p_3, p_4\}$ of properties; (b) Subgraphs involving three properties from p_1, p_2, p_3 , and p_4 ; (c) Subgraphs around two properties from p_1, p_2, p_3 , and p_4 .

star pattern, shown in Figure 1c, extracted from the RDF graph in Figure 1a; note that $?x$ is a variable whose instantiations correspond to constants in the RDF graph. In these RDF subgraphs, the properties p_1, p_2 , and p_3 , and the corresponding objects e_1, e_2 , and e_3 , respectively, are the same, whereas the entities c_1, c_2, c_3 , and c_4 are different. This indicates that the star pattern is a frequent star pattern, i.e., several entities c_1, c_2, c_3 , and c_4 instantiate the star pattern. Thus, several entities are related to the same objects, even not all the properties of the class are involved in frequent star patterns. A frequent star pattern comprising the entities c_1, c_2, c_3 , and c_4 is illustrated in Figure 1c, where the node $?x$ represents the entities c_1, c_2, c_3 , and c_4 of class C in the RDF graph in Figure 1a. gSpan [30] solves the problem of identifying the frequent subgraphs that involve same subject entities related to the same object values using a set of properties. However, our approach requires the identification of frequent star patterns, where each star pattern—with a subject variable—involves different subject entities related to the same object values using a set of properties. Figures 2a, 2b, and 2c show some of the subgraphs extracted by gSpan involving entities c_1 and c_4 , and the sets of properties containing four, three, and two properties, respectively, from the RDF graphs in Figure 1a. gSpan exhaustively

enumerates the frequent subgraphs; thus, finding frequent star patterns requires an exhaustive search over the generated frequent subgraphs. In this work, we exploit the RDF model and propose a technique that allows for transforming an RDF graph G into another RDF graph G' where the number of frequent star patterns is minimized. The graph G' includes all the nodes from G but additionally, G' comprises nodes that represent *factorized entities*—like the one in Figure 4c.

3 Related Work

Database and Semantic Web communities have proposed several representations to speed up processing over the large amounts of data represented using relational and RDF data models [1, 2, 3, 14, 16, 17, 20, 23, 32]. These compression approaches can be categorized into compression techniques for relational and RDF graph data models. Relational data model approaches [1, 32] efficiently store very large datasets in column-oriented stores. Approaches [3, 14, 16, 17, 20, 21, 23, 31] target the efficient storage of RDF graph data. Furthermore, several frequent pattern mining algorithms [11, 30] extract frequent isomorphic graph patterns from a graph.

3.1 Data Compression for Relational Data Models

Column-oriented databases [27, 32] store each attribute in a separate column such that successive values of the attribute are accumulated consecutively on the disk. This improves the query processing when the values of some of the columns are required to process the query. The column oriented data storage opens a number of opportunities to apply compression techniques more naturally over the multiple values of the same type. Compression approach proposed by Abadi et al. [1] compress each column in C-store [27] using one of the methods like Null Suppression, Dictionary Encoding, Run-length Encoding, Bit-Vector Encoding or Lempel-Ziv [25, 29]. Zukowski et al. [32] focus on improving bad CPU/-cache performance caused by the compression techniques involving if-then-else statements in the code, e.g., Null Suppression, Run-length Encoding, and does not take advantage of the super-scalar properties, e.g., pipe-lining the processes, in the modern CPUs. Zukowski et al. propose three compression methods i.e., PFOR, PFOR-DELTA, and PDICT. These compression solutions are exploited by column-oriented stores using the decomposition storage model [10], where n -array relations are decomposed into n binary relations. Each binary relation consists of one attribute values and the corresponding surrogate keys. In this model, two copies of data are stored increasing the data storage requirements. Further, for each attribute a copy of the corresponding duplicated surrogate key is required resulting in an increase of the storage by a factor of two. Moreover, various compression techniques for a large number of unique values, i.e., subject entities, are hard to implement. Our approach generates a factorized graph where entities matching a frequent star pattern are represented by a surrogate entity of the corresponding compact RDF molecule. These compact graph representations replace repeated properties and corresponding objects with properties and objects in the compact RDF molecules, hence, improve the storage space requirements for the decomposition storage model [10].

3.2 Data Compression for the RDF Data Model

Meier et al. [20] propose a user-specific minimization technique based on Datalog rules to remove the RDF triples from a given RDF graph. Similarly, Pichler et al. [23] study the RDF redundancy elimination in the presence of rules, constraints, and queries specified by users. These two approaches are user specific and require human input for compressing the ever growing RDF graphs. A scalable lossless RDF compression technique, proposed by Joshi et al. [16], automatically generates decompression rules. The rules are used to split the RDF datasets into an active dataset containing compressed triples, and a dormant dataset consisting of uncompressed RDF triples. This technique requires the overhead of decompression over the compressed data to access the information initially represented in datasets. A factorized representation of RDF graphs is presented by Karim et al. [17], where repeated observation values are represented only once. This approach reduces the number of RDF triples in the observational data, which is semantically described using the Semantic Sensor Network (SSN) Ontology [9]. We propose an approach to automatically identify frequent star patterns in RDF graphs described using any ontology. Further, we devise factorized graphical representations of RDF graphs which do not require data decompression to perform data management tasks. Fernández et al. [14] present a binary RDF representation format consisting of a Header, a Dictionary and a Triple component containing RDF metadata, RDF terms catalog, and compactly encoded RDF triples, respectively. Pan et al. [21] propose RDF compression based on graph patterns, which reduces the number of RDF triples and then generates compact binary representations of the reduced triples. The compression technique k^2 -triples presented by Álvarez-García et al. [3] exploits the two dimensional k^2 -trees structure, proposed by Barisaboa et al. [8], to distribute the compact triples obtained by Header-Dictionary-Triples partitioning [14]. These approaches are able to effectively reduce redundancies in RDF graphs, and provide effective techniques for RDF graph compression. However, customized

engines are required to perform query processing over the compressed RDF graphs, and decompression techniques are needed during data management. We devise factorization techniques that use semantics encoded in RDF data and compactly represent RDF triples, reduce redundancy, and facilitate data management tasks without requiring any decompression or a customized engine.

3.3 Graph Mining Techniques

The problem of frequent pattern mining involves finding subgraphs, from a graph, that have frequency above a given threshold. gSpan [30] exploits the depth first search (DFS) to mine frequent patterns. gSpan maps a graph to a DFS code representing the edges sequence. Several DFS codes can be generated for a single graph. These DFS codes are ordered lexicographically based on the edge labels and the order of nodes being visited. From these ordered DFS codes the minimum DFS codes are selected to build the DFS tree. DFS over a code tree discovers all the minimum DFS codes of frequent patterns. GRAMI [11] mines frequent patterns and finds only the minimal set of instances that satisfy the given frequency threshold. GRAMI stores the templates of frequent patterns instead of storing their appearances. This avoids the creation and storage of all appearances of patterns. For frequency evaluation, GRAMI maps the frequent patterns mining problem to constraint satisfaction problem (CSP), which is represented by a tuple; (a) an ordered set of variables representing nodes, (b) a set of domains of variables in (a), and (c) a set of constraints between these variables. Two subgraphs patterns are isomorphic if the variables in corresponding CSP tuple have different values from the domains, however, nodes and edge labels are the same. Notwithstanding these frequent pattern mining approaches are able to identify the frequent isomorphic graph patterns, extracting frequent star patterns, which involve different subject nodes related with same objects nodes using same set of edge labels, requires an exhaustive search over the identified frequent patterns. It is important to highlight that although these approaches effectively mine subgraph patterns, they are not able to iden-

tify patterns where one node is a variable. Contrary, our approach searches for star patterns and is able to detect the ones with highest instantiations.

4 RDF Graph Factorization Approach

We introduce important preliminary definitions, and then formally define the problem of detecting frequent star patterns and compacting them in an RDF graph.

4.1 Preliminaries

Our approach is based on the RDF data model building on RDF triples.

Definition 4.1 (RDF triple [4]). Let \mathbf{I} , \mathbf{B} , \mathbf{L} be disjoint infinite sets of URIs, blank nodes, and literals, respectively. A tuple $(s \ p \ o) \in (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L})$ is an RDF triple, where s is the subject, p is the property, and o is the object.

A set of RDF triples is called RDF dataset (or knowledge graph) and can also be viewed as a graph. Thus, in Figure 1a, the edge $(c_1 \ type \ C)$ represents an *RDF triple*, where entity c_1 corresponds to subject, *type* and C represent a property and an object, respectively; there are nineteen more *RDF triples*.

Definition 4.2 (RDF Graph). An RDF graph $G = (V, E, L)$ is a labeled directed graph where nodes represent entities or objects, while labels stand for properties:

- An RDF triple $(s \ p \ o) \in E$, corresponds to an edge in E from node s to node o ; p is the label of the edge and denote the property that relates both nodes;
- $s, o \in V$, s corresponds to a subject and o corresponds to an object; and
- $p \in L$, is an edge label corresponding to a property.

Definition 4.3 (RDF Molecule [13]). An RDF molecule RM is a set of RDF triples that share the same subject, i.e., $RM = (s\ p_1\ o_1), (s\ p_2\ o_2), \dots, (s\ p_n\ o_n)$.

Figure 1b presents four RDF molecules around the subjects c_1, c_2, c_3 , and c_4 of class C . In the RDF molecule around subject c_1 all the RDF triples describe c_1 using properties p_1, p_2 , and p_3 . Similarly, RDF triples in each of the other RDF molecules describe the subjects c_2, c_3 , and c_4 using properties p_1, p_2 , and p_3 .

4.2 Problem Statement

Star patterns denote graph patterns covering RDF molecules:

Definition 4.4 (Star Pattern). Given is an RDF graph $G = (V, E, L)$, a class C in E and a set of properties $SP = \{p_1, p_2, \dots, p_n\}$ such that C is the domain of all the properties in SP . Let entities o_1, o_2, \dots, o_n be the objects of the properties p_1, p_2, \dots, p_n , respectively. Let $?s$ be a variable. A star pattern of C over the properties p_1, p_2, \dots, p_n and objects o_1, o_2, \dots, o_n corresponds to a graph pattern composed of the conjunction of triple patterns: $(?s\ p_1\ o_1), (?s\ p_2\ o_2), \dots, (?s\ p_n\ o_n)$.

Figure 1c shows a star pattern composed of three triple patterns containing properties p_1, p_2 , and p_3 and the corresponding objects e_1, e_2 , and e_3 , respectively. The entities c_1, c_2, c_3 , and c_4 of class C in the RDF graph in Figure 1a match the star pattern. The variable $?x$ is the subject of the triple patterns referring to the entities matching the star pattern.

Definition 4.5 (Class Multiplicity). Given an RDF graph $G = (V, E, L)$, a class C in E and a set of properties $SP = \{p_1, p_2, \dots, p_n\}$ such that C is the domain of all the properties in set SP of properties. Let entities o_1, o_2, \dots, o_n be objects of the properties p_1, p_2, \dots, p_n , respectively. The multiplicity of o_1, o_2, \dots, o_n in G , $M(o_1, o_2, \dots, o_n|G)$ is defined as the number of different entities in C that match a star pattern having the same objects o_1, o_2, \dots, o_n in the properties p_1, p_2, \dots, p_n . Entities s correspond

to instantiations of the subject variable in the star pattern.

$$M(o_1, o_2, \dots, o_n|G) = |\{s \mid (s\ \text{type}\ C) \in G, \\ (s\ p_1\ o_1) \in G, (s\ p_2\ o_2) \in G, \\ \dots, (s\ p_n\ o_n) \in G\}|$$

In the RDF graph in Figure 1a, the multiplicity of the objects e_1, e_2 and e_3 , given the set $\{p_1, p_2, p_3\}$ of properties, is 4, because there are four instantiations of the subject variable. Similarly, the multiplicity of objects e_4, e_5 and e_6 , in the set $\{p_4\}$ of properties is 1 and 2.

Definition 4.6 (Class Multiplicity Inverse). Given class C , a set $SP = \{p_1, p_2, \dots, p_n\}$ of properties and corresponding objects o_1, o_2, \dots, o_n , the multiplicity inverse of o_1, o_2, \dots, o_n in G , denoted $MI(o_1, o_2, \dots, o_n|G)$, is:

$$MI(o_1, o_2, \dots, o_n|G) = 1/M(o_1, o_2, \dots, o_n|G)$$

In the RDF graph in Figure 1a, the *class multiplicity inverse* of the objects e_1, e_2 , and e_3 , given the set $\{p_1, p_2, p_3\}$ of properties, is $\frac{1}{4}$. The multiplicity inverse of objects e_4, e_5 , and e_6 in the set $\{p_4\}$ of properties is $\frac{1}{1}$ and $\frac{1}{2}$.

Definition 4.7 (Multiplicity of Star Patterns). Given a class C in an RDF graph G with properties $SP = \{p_1, p_2, \dots, p_n\}$. The multiplicity of the star patterns in C over SP , $AMI_G(p_1, p_2, \dots, p_n|C)$, is defined as follows:

$$AMI_G(p_1, p_2, \dots, p_n|C) = [f'_{\forall s \in C}(\{MI(o_1, o_2, \dots, o_n|G) \mid (s\ \text{type}\ C) \in G, (s\ p_1\ o_1) \in G, \\ (s\ p_2\ o_2) \in G, \dots, (s\ p_n\ o_n) \in G\})]$$

where $f'(\cdot)$ is an aggregation (e.g., summation) function.

In the RDF graph in Figure 1a, the *multiplicity of the star patterns* of C over the set $\{p_1, p_2, p_3\}$ of properties is $\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4} = 1$, which is obtained by summing up the class multiplicity inverse of the objects e_1, e_2 , and e_3 given the set $\{p_1, p_2, p_3\}$ of properties, for each entity c_1, c_2, c_3 , and c_4 of class

C matching the star pattern. Similarly, the *multiplicity of the star patterns* of class C over the set $\{p_4\}$ is $\frac{1}{2} + \frac{1}{2} + \frac{1}{1} + \frac{1}{1} = 3$ in the RDF graph, and is obtained by summing up the individual class multiplicity inverse of objects e_4 , e_5 , and e_6 given the set $\{p_4\}$, for each of the entities c_1 , c_2 , c_3 , and c_4 of class C that map the corresponding star patterns. The *multiplicity of the star patterns* over a set of properties corresponds to the number of star patterns composed of the set of properties and the corresponding objects. The problem of frequent star patterns detection is defined next, the solutions correspond to frequent star patterns. We define the frequent star patterns detection problem as the minimization of connections between a class instances and values linked through the properties. To find the minimum number of edges over the properties in a class, the sum of the number of edges in the star patterns over a set of properties and the number of edges between the class entities and the properties that are not involved in the star patterns is computed.

Definition 4.8 (FSP Detection Problem). Given an RDF graph $G = (V, E, L)$ and a class C in G with set of properties S and number of instances $AM_G(C)$. The problem of *Frequent Star Patterns Detection* (FSP Detection) is to find a subset SP of S such that the star patterns SGP of C over SP corresponds to *frequent star patterns*, i.e., $\#Edges(SP, C, G)$ is minimized:

$$\arg \min_{SP \subseteq S} \underbrace{\{AMI_G(SP|C) * (|SP| + 1) + AM_G(C) * (|S - SP|)\}}_{\#Edges(SP, C, G)} \quad (1)$$

Figure 3 illustrates the problem of detecting frequent star patterns from the RDF graph in Figure 1a. Figure 3a presents three star patterns $AMI_G(SS|C)$ over the set of properties p_1 , p_2 , p_3 , and p_4 , and 15 edges in $\#Edges(SS, C, G)$. However, only one star pattern $AMI_G(SS'|C)$ over the set of properties p_1 , p_2 , and p_3 exists in Figure 3b. A small value of $\#Edges(SS', C, G)$ i.e., eight, shows a subgraph over SS' that is represented by only one star pattern with more instantiations than the star patterns for SS , i.e., it is a frequent star pattern. Thus, the set of properties SP where $\#Edges(SP, C, G)$ is minimal, encloses a subgraph with the minimal number of star

patterns which have the maximal number of instantiations; additionally, these star patterns are the ones with the greater number of properties. Figure 3c depicts the factorized RDF graph where this frequent star pattern has been replaced with a compact RDF molecule on a surrogate entity cM ; this factorization reduces the size of the original RDF graph.

Theorem 4.1. Given an RDF graph G , a class C in G , and non-empty sets of properties S , SP , and SP' of C such that $SP' \subset SP \subset S$. If $\#Edges(SP', C, G) > \#Edges(SP, C, G)$, then $\forall SP'' \subset SP'$, $\#Edges(SP'', C, G) \geq \#Edges(SP, C, G)$.

Proof. By contradiction. Suppose $\#Edges(SP'', C, G) < \#Edges(SP, C, G)$. From $\#Edges(SP', C, G) > \#Edges(SP, C, G)$ and $SP' \subset SP \subset S$, it can be inferred that $AMI_G(SP|C) < AM_G(C)$, $AMI_G(SP'|C) < AM_G(C)$, $|SP''| < |SP'| < |SP| < |S|$, $|SP - SP''| \geq 2$, and $AMI_G(SP''|C) < AM_G(C)$. Considering these inequalities in $\#Edges(SP'', C, G)$ and $\#Edges(SP, C, G)$, we can demonstrate that $\#Edges(SP'', C, G)$ is at least greater than $\#Edges(SP, C, G)$ in $2 * AM_G(C)$, contradicting, thus, $\#Edges(SP'', C, G) < \#Edges(SP, C, G)$. \square

Definition 4.9 (A Compact RDF Molecule). Given a star pattern SGP of a class C over the properties p_1, p_2, \dots, p_n and objects o_1, o_2, \dots, o_n . Given a surrogate entity sg of type C . A compact RDF molecule for SGP is an RDF molecule composed of RDF triples $(sg \ p_1 \ o_1), (sg \ p_2 \ o_2), \dots, (sg \ p_n \ o_n)$.

Figure 4c shows a compact RDF molecule that instantiates the star pattern presented in Figure 1c, which is composed of the properties p_1 , p_2 , and p_3 and the corresponding objects e_1 , e_2 , and e_3 , respectively. The surrogate entity cM in the compact RDF molecule, represents the entities c_1 , c_2 , c_3 , and c_4 of type C matching the star pattern, as shown in Figure 1b.

Definition 4.10 (The RDF-F Problem). Given an RDF graph $G = (V, E, L)$ and a set of properties

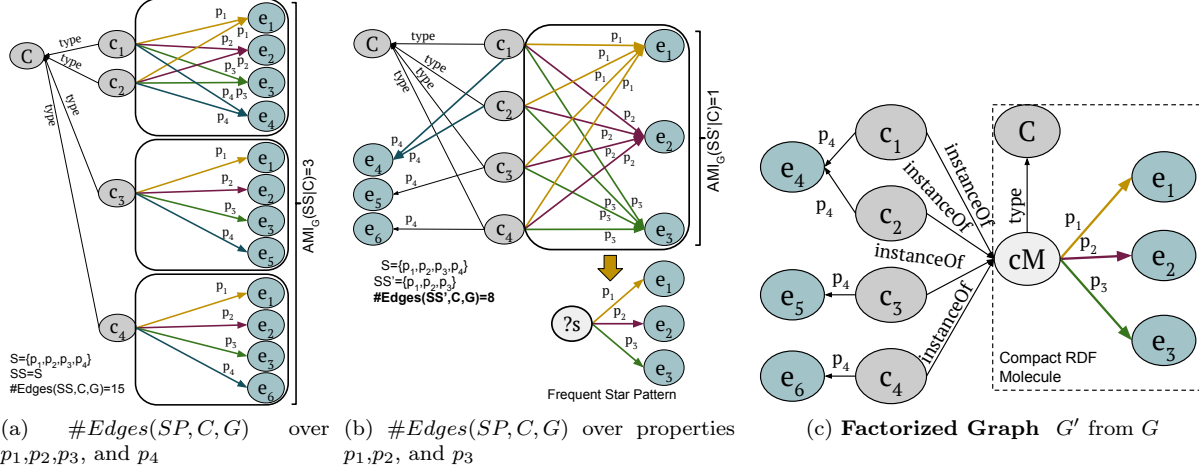


Figure 3: **The Frequent Star Patterns Detection Problem.** Properties involved in frequent star patterns. (a) Stars patterns over the set $SS = \{p_1, p_2, p_3, p_4\}$ of properties in class C require three surrogate entities and $\#Edges(SS, C, G)$ are 15; (b) Star patterns over the set $SS' = \{p_1, p_2, p_3\}$ of properties in class C require one surrogate entity and $\#Edges(SS', C, G)$ are eight; (c) A factorized RDF graph G' of G composed of compact RDF molecule with a surrogate entity cM .

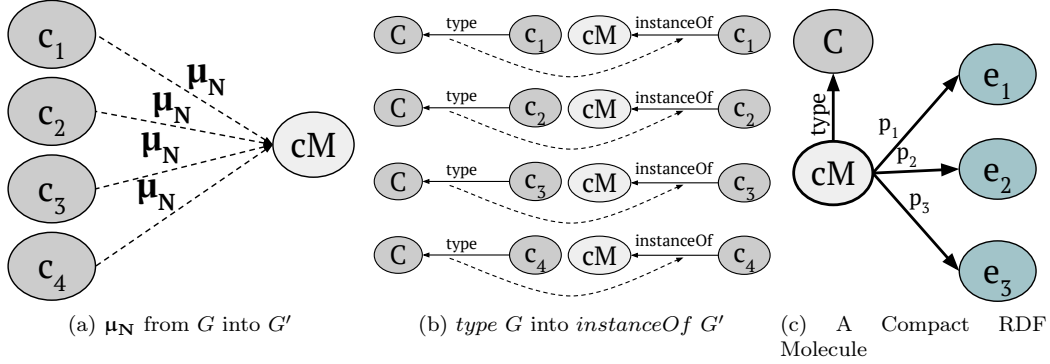


Figure 4: **The RDF Graph Factorization Problem.** Factorization of RDF graph G into G' . (a) Entity mappings μ_N from the RDF graph G in 1a to the surrogate entity cM in G' ; (b) Transformation of property $type$ from G to G' ; (c) A compact RDF molecule for the frequent star pattern over the properties p_1, p_2 , and p_3 .

SP , the problem of *RDF factorization (RDF-F)* corresponds to finding a *factorized RDF graph* of G , $G' = (V', E', L')$, where the following hold:

- Entities in G are preserved in G' , i.e., $V \subseteq V'$.
- For each entity s_i in V that corresponds to an in-

stantiation of the variable of a frequent star pattern SGP of a class C over the set SP in G , there is an entity s_{SGP} in V' that corresponds to the surrogate entity of the compact RDF molecule of SGP . Formally, there is a partial mapping $\mu_N: V \rightarrow V'$:

- Instances of the frequent star pattern SGP are mapped to the surrogate entity of the star pattern, i.e., $\mu_N(s_i) = s_{SGP}$.
- The mapping μ_N is not defined for the rest of the entities that do not instantiate a frequent star pattern in G .

- For each RDF triple t in $(s p o)$ in E :
 - If $\mu_N(s)$ is defined and C_s is the type of s , and p is *type*, then the triples $(s \text{ instanceOf } \mu_N(s))$, $(\mu_N(s) \text{ type } C_s)$ belong to E' .
 - If $\mu_N(s)$ is defined and C_s is the type of s , and $p \in SP$, then the triples $(\mu_N(s) p o)$ belong to E' .
 - Otherwise, the RDF triple t is preserved in E' .

Consider RDF graphs G and G' shown in Figures 1a and 3c, respectively. Figure 4a depicts a map μ_N that assigns entities c_1, c_2, c_3 , and c_4 of class C in G to the surrogate entity cM in G' . Further, entities $c_1, c_2, c_3, c_4, C, e_1, e_2, e_3, e_4, e_5$, and e_6 are preserved in G' . Moreover, the edge labeled with property p_1 in G , i.e., $(c_1 p_1 e_1)$ is presented with edges $(c_1 \text{ instanceOf } cM)$, $(cM p_1 e_1)$ and $(cM \text{ type } C)$ in G' ; similarly, edges labeled with properties p_2 and p_3 in G are represented in G' . Figure 4b shows the transformations of the connections between entities c_1, c_2, c_3 , and c_4 and the class C using labeled edges annotated with property *type*, with the connections relating the entities c_1, c_2, c_3 , and c_4 to the corresponding surrogate entity cM using the property *instanceOf*.

Definition 4.11 (Axioms for InstanceOf). The property *instanceOf* is a functional property defined as follows:

- If $(s_i \text{ instanceOf } sg)$ and $(sg \text{ type } C)$ then $(s_i \text{ type } C)$.
- If $(s_i \text{ instanceOf } sg)$ and $(sg p_j o_k)$ then $(s_i p_j o_k)$.

These two axioms enable to represent implicitly, all the knowledge encoded in the edges from an original

RDF graph that are removed during the factorization process. They are utilized during query processing to rewrite queries over the original RDF graph into queries against the factorized RDF graph.

4.3 FSP Detection Approach

Algorithm 1 E.FSP Algorithm

Input: A dictionary *subgraphsDict* of subgraphs over the subsets of properties in S , A set S of properties of class C .

Output: Frequent star patterns *fsp*, A set SP of properties

```

1:  $fsp \leftarrow \square$ ,  $SP \leftarrow \emptyset$ ,  $minEdges \leftarrow 0$ ,  $subsetCard \leftarrow |S|$ 
2: while  $subsetCard \geq 2$  do
3:    $propSets \leftarrow getSubsetsOf(S, subsetCard)$ 
4:   for  $SP \in propSets$  do
5:      $subgraphs \leftarrow subgraphsDict[SP]$ 
6:      $totalEdges \leftarrow countEdges(subgraphs)$ 
7:     if  $minEdges == 0$  then
8:        $minEdges \leftarrow totalEdges$ 
9:        $fsp \leftarrow subgraphs$ 
10:       $bestSP \leftarrow SP$ 
11:     else if  $totalEdges < minEdges$  then
12:        $minEdges \leftarrow totalEdges$ 
13:        $fsp \leftarrow subgraphs$ 
14:        $bestSP \leftarrow SP$ 
15:     end if
16:   end for
17:    $subsetCard \leftarrow subsetCard - 1$ 
18: end while
19:  $SP \leftarrow bestSP$ 
20: return  $fsp, SP$ 

```

To solve the *FSP detection* problem, we propose two algorithms that perform iterations over frequent patterns involving different sets of properties sets of a class C in an RDF graph G , and the class entities. *E.FSP*, presented in Algorithm 1, resorts to a frequent pattern mining algorithm like gSpan. *E.FSP* exploits breadth first search technique to exhaustively traverse the search space of frequent patterns generated by the frequent pattern mining algo-

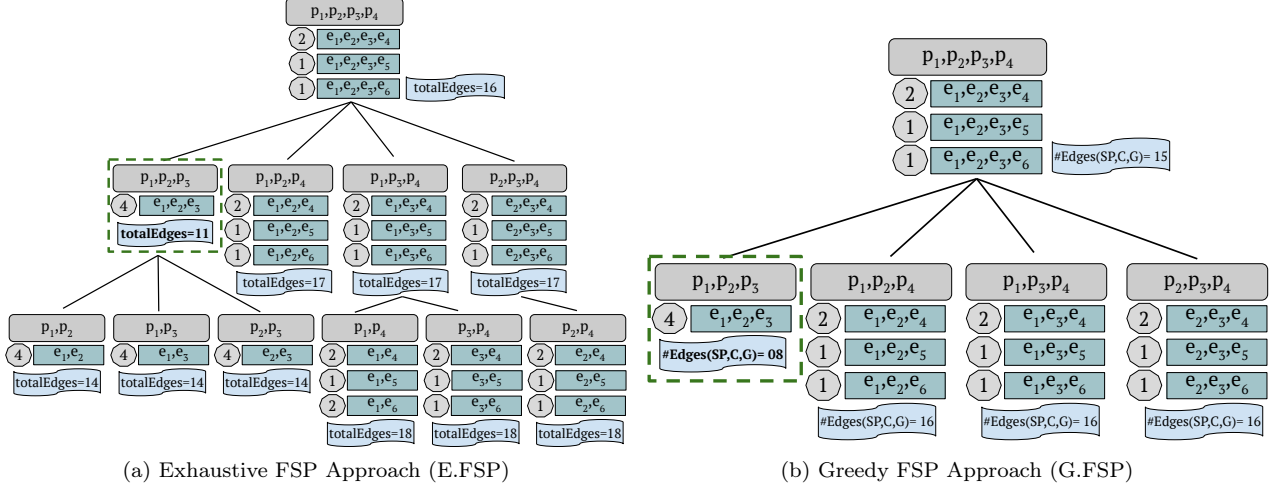


Figure 5: **Frequent Star Patterns Detection.** *E.FSP* and *G.FSP* iterate over the star patterns in the RDF graph in Figure 1a to detect the frequent star patterns. (a) *E.FSP* exhaustively iterates over the whole search space of frequent patterns; (c) *G.FSP* iterates the search space without generating all the star patterns.

rithm, and always finds the best frequent star patterns. Figure 5a illustrates the iterations performed by *E.FSP* to find the frequent star patterns in the RDF graph in Figure 1a. *E.FSP* receives a dictionary *subgraphsDict* of all the subgraphs over the subsets of the set S of properties in the class C in an RDF graph G . The keys of the dictionary *subgraphsDict* are the combination of properties in the subsets of S , and the dictionary values are the subgraphs involving the properties from the corresponding keys. *E.FSP* generates frequent star patterns and a set of properties involved in the frequent star patterns. *E.FSP* initializes the variables *fsp*, *SP*, *minEdges*, and *subsetCard* in line 1. The variables *minEdges* and *subsetCard* are initialized with values 0 and cardinality of S , respectively. From lines 2-18, *E.FSP* iterates over all the subgraphs involving two or more properties to find the frequent star patterns. In Figure 5a, *E.FSP* starts iterations with the set of properties $SP = \{p_1, p_2, p_3, p_4\}$, and the subgraphs involving the properties in subsets of SP , where the cardinality of subsets is equal to the cardinality of S , i.e., four (line 3). The generated subset contains all the properties in SP , i.e., $\{p_1, p_2, p_3, p_4\}$, and the

total number of edges *totalEdges* in SP is computed i.e., 16 (line 5-6). Since *minEdges* are 0, therefore, the value 16 of *totalEdges* is assigned to *minEdges*, subgraphs over $SP = \{p_1, p_2, p_3, p_4\}$ and SP are assigned to *fsp* and *bestSP*, respectively (line 7-10). At line 17, the subset size *subsetSize* is reduced by one in order to generate the subsets of properties of S i.e., three. The subsets $\{p_1, p_2, p_4\}$, $\{p_1, p_3, p_4\}$, and $\{p_2, p_3, p_4\}$, of cardinality three, generate more number of edges i.e., value of *totalEdges* is 17, than the minimum number of edges *minEdges*, i.e., 16, and are not selected as the best sets of properties. However, the subgraphs over the subset $\{p_1, p_2, p_3\}$ contain 11 number of triples, which is less than 16 the value of *minEdges*. Therefore, *E.FSP* selects $\{p_1, p_2, p_3\}$ as the best set of properties and the corresponding subgraphs as the frequent star patterns (line 11-15). Once all the subsets SP of S with cardinality three, are evaluated, the value of *subsetCard* is reduced by one i.e., two, and the subsets of cardinality two are evaluated in the next iteration. Figure 5a presents that all the subsets of cardinality two generate larger values, i.e., 14 and 18, for *totalEdges* than the value

11 for *minEdges*. Therefore, none of the subsets of properties of cardinality two contains the frequent star patterns. Further, all the subsets of cardinality greater or equal to two have been evaluated, *E.FSP* stops and returns $\{p_1, p_2, p_3\}$ as the best set of properties and the corresponding subgraphs as the frequent star patterns (line 19-20).

G.FSP, presented in Algorithm 2, adopts a greedy algorithm to traverse the search space without generating all the frequent patterns. *G.FSP* starts iterations using a set *SP* of properties containing all the properties in *S* of a class *C* in an RDF graph *G*. *G.FSP* computes the value of Formula 1 for *SP* and iterates over the subsets *SP'* of cardinality one less the cardinality of *SP* and computes Formula 1 for each of subsets *SP'*. A property subset *SP'* with a smaller formula value than the formula value of *SP*, is selected as the best set of properties in that iteration, and is used in the next iteration to check the subsets of cardinality one less the cardinality of the selected set of properties. The iterations are performed until the cardinality of the selected subset of properties is less than two. Based on the property presented in Theorem 4.1, *G.FSP* stops, if none of the subsets *SP'* generates less value for formula than the formula value of *SP*. In addition, *G.FSP* stops whenever the cardinality of the set of properties is less than two, or the multiplicity of star patterns $AMI_G(SP|C)$ is one. *G.FSP* receives a set *S* of properties in class *C* in an RDF graph *G*, and a list *starList* of star patterns involving properties in *S*. *G.FSP* returns frequent star patterns *fsp* and a set of properties *SP* involved in the frequent star patterns. Figure 5b shows the iterations performed by *G.FSP* to detect the frequent star patterns in the RDF graph in Figure 1a. *G.FSP* initializes all the variables at line 1, where *SP* is assigned the set *S* of properties for the first iteration i.e., $SP = \{p_1, p_2, p_3, p_4\}$. In lines 2-29, *G.FSP* iterates over the subsets of *SP* to find the frequent star patterns based on the criteria in Formula 1. The cardinality value four of *SP* is greater than two (line 3), and $AMI_G(SP|C)$ is not equal to one (line 4-7), therefore, *G.FSP* computes the value of $\#Edges(SP, C, G)$ of *SP* i.e., 15 (line 8). In lines 9-25, *G.FSP* iterates over the subsets of *SP* of cardinality one less the cardinality of *SP*

Algorithm 2 G.FSP Algorithm

Input: A set *S* of properties of class *C* in *G*, and a list *starList* of star patterns over properties in *S*.

Output: Frequent star patterns *fsp*, A set *SP* of properties.

```

1: fsp  $\leftarrow []$ , starList'  $\leftarrow []$ , SP  $\leftarrow S$ , SP'  $\leftarrow \emptyset$ ,
   fValue  $\leftarrow fValue' \leftarrow 0$ 
2: repeat
3:   if  $|SP| \geq 2$  then
4:     if  $AMI_G(SP|C) == 1$  then
5:       fsp  $\leftarrow starList$ 
6:       return fsp, SP
7:     else
8:       fValue  $\leftarrow \#Edges(SP, C, G)$ 
9:       for p  $\in SP$  do
10:        SP'  $\leftarrow SP - \{p\}$ 
11:        if  $|SP'| \geq 2$  then
12:          Create starList' over SP' using
            starList
13:          value  $\leftarrow \#Edges(SP', C, G)$ 
14:          if  $AMI_G(SP'|C) == 1$  then
15:            fValue'  $\leftarrow value$ 
16:            bestSP  $\leftarrow SP'$ 
17:            bestSList  $\leftarrow starList'$ 
18:            break
19:          else if value < fValue' then
20:            fValue'  $\leftarrow value$ 
21:            bestSP  $\leftarrow SP'$ 
22:            bestSList  $\leftarrow starList'$ 
23:          end if
24:        end if
25:      end for
26:    end if
27:  end if
28:  starList  $\leftarrow bestSList$ , SP  $\leftarrow bestSP$ 
29: until fValue' > fValue
30: fsp  $\leftarrow starList$ 
31: return fsp, SP

```

to find the best set of properties for the next iteration. At line 10, a property p is removed from SP to generate a subset SP' e.g., by removing p_1 a subset $SP' = \{p_2, p_3, p_4\}$ is generated. Since the cardinality of SP' is more than two, therefore, a star list $starList'$, representing the star patterns over SP' , is created using $starList$ (line 12-13). The value of $\#Edges(SP', C, G)$ for SP' is computed i.e., 16 (line 13). For SP' , $AMI_G(SP'|C)$ is not one, and the value 16 of $\#Edges(SP', C, G)$ for SP' is not less than the value 15 of $\#Edges(SP, C, G)$ for SP , therefore, the star patterns over $SP' = \{p_2, p_3, p_4\}$ do not involve frequent star patterns and SP' is not a best candidate for the next iteration. Similarly, the property subsets $\{p_1, p_3, p_4\}$ and $\{p_1, p_2, p_4\}$, generated from SP by removing p_2 and p_3 , respectively, give a higher value 16 for $\#Edges(SP', C, G)$ and the star patterns over these set of properties are not better than the star patterns over SP . However, $SP' = \{p_1, p_2, p_3\}$, generated from SP by removing p_4 , gives one star pattern, therefore, the star pattern involving properties in SP' is returned as the frequent star pattern without performing more iteration (line 14-18). In case, the set SP' of properties is involved in more than star patterns and the formula value of SP' smaller than the value of SP , then SP' is selected for the next iteration (line 19-23). $G.FSP$ stops and no further iterations are performed if none of the subsets SP' of SP generates a smaller value for $\#Edges(SP', C, G)$ than $\#Edges(SP, C, G)$. $G.FSP$ returns the star patterns involving SP , with a minimum value for $\#Edges(SP, C, G)$, as the frequent star patterns, and SP as the best set of properties. $E.FSP$ and $G.FSP$ work under the following assumptions: (a) all RDF molecules are complete, i.e., all class entities have values for all the properties, (b) all the properties are functional. In addition to these assumptions, $G.FSP$ has one more assumption: (c) if there are ties while deciding between the sets of properties, only one will be selected. Complexity of $E.FSP$ is exponential, i.e., 2^n . $G.FSP$ adopts a Greedy approach and prunes the search space by selecting only the best set of properties during each iteration until the stop condition is met, i.e., no better set of properties with a minimum formula value can be found. In the worst case, the computational complexity of

$G.FSP$ is $\sum_{i=0}^n (n-i) = \frac{n(n+1)}{2}$, where n is the cardinality of the input set of properties. The complexity of $G.FSP$ grows linearly with the increase in the size of the input set of properties.

4.4 A Factorization Approach

We present a solution to the problem of factorizing RDF graphs describing data using ontologies. A sketch of the proposed method is presented in Algorithm 3. The algorithm receives an RDF graph $G = (V, E, L)$, a class C , and a set SP' of properties from $E.FSP$ or $G.FSP$, and generates a factorized RDF graph $G' = (V', E', L')$, and the entity mappings μ_N from the entities of class C in V in RDF graph G to the surrogate entities in V' in RDF graph G' . The algorithm initializes the set of mappings μ_N , the set of nodes V' , the set of labeled edges E' and the set of edge labels (properties) L' of the factorized RDF graph G' (line 1). For all the entities of C related to the same objects o_1, o_2, \dots, o_n using edges annotated with properties p_1, p_2, \dots, p_n in SP' , the algorithm creates a surrogate entity sg for the corresponding compact RDF molecule in G' (lines 2-3). In lines 4-6, the algorithm maps all the entities, that are related to o_1, o_2, \dots, o_n using properties p_1, p_2, \dots, p_n in G , to the surrogate entity in μ_N . Once all the mappings of the entities of C in G to the corresponding surrogate entities in G' are in μ_N , the factorized RDF graph G' is created using μ_N (lines 8-29). For each RDF triple $(s \ p \ o)$ in E , if entity mapping $\mu_N(s)$ is defined, then a compact RDF molecule is created. If p is *type*, then the new edges $(s \ instance \ \mu_N(s))$, and $(\mu_N(s) \ p \ o)$ are added to G' along with entities s , o and the mapped surrogate entity of s , and the edge labels p and *instanceOf* (lines 11-14). If p is in SP , the new edge $(\mu_N(s) \ p \ o)$, and entities s , o and the edge label p are added to G' (lines 15-18). If entity mapping $\mu_N(s)$ are defined, however, p is not in SP , or p is not *type*, then the edge $(s \ p \ o)$ is added to G' along with the corresponding nodes and the edge label (lines 19-23). If entity mapping $\mu_N(s)$ is not defined, then the edge $(s \ p \ o)$ and the nodes s and o , and edge label p are added to G' (lines 24-28).

Figure 6 depicts the transformations for the set $\{p_1, p_2,$

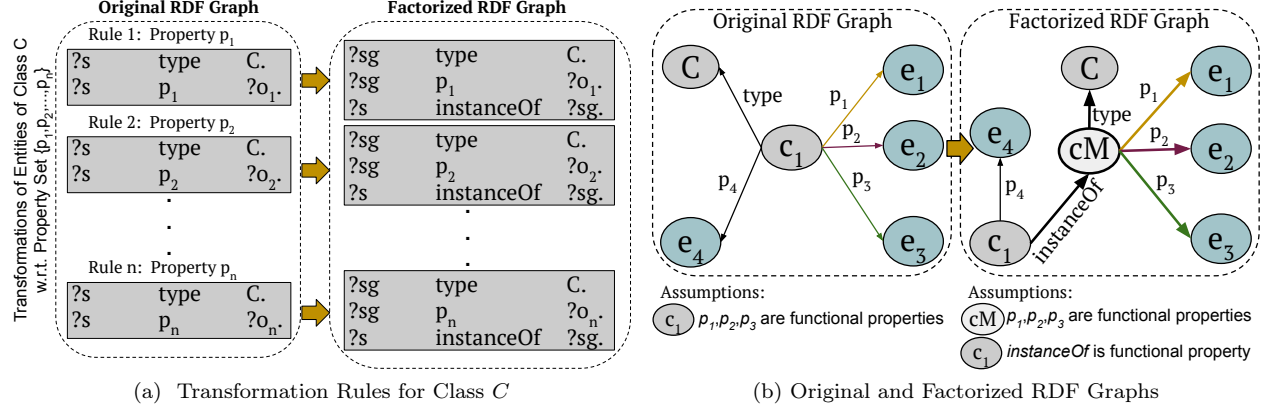


Figure 6: **Transformations in RDF Graph.** Transformation rules preserved between original and factorized RDF graphs. (a) Transformation rules over the properties p_1, p_2, \dots, p_n ; (b) Portions of RDF graphs (original and factorized). Nodes and edges added to create the factorized RDF graph, are highlighted in bold.

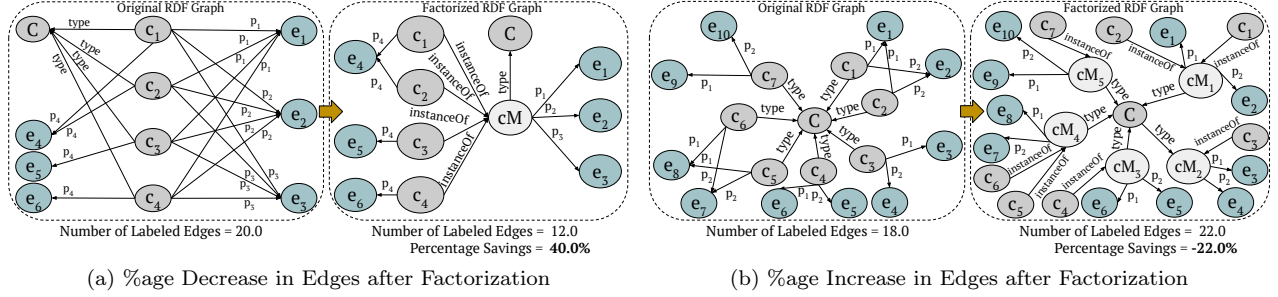


Figure 7: **RDF Graph Factorization Overhead.** Factorization of RDF graphs is not worthy in all cases. (a) Entities of class C in the original RDF graph match the frequent star pattern over the properties p_1, p_2 , and p_3 ; (b) few entities match each star pattern over p_1 and p_2 causing factorization overhead.

$\dots, p_n\}$ of properties performed in an RDF graph whenever a corresponding factorized RDF graph is created. Figure 6a presents transformation rules; one rule for each property in $\{p_1, p_2, \dots, p_n\}$ of class C . Each rule states how the labeled edges associated with a C in an original RDF graph are transformed into the edges in the factorized graph. Rule 1 asserts that the relation between an entity s of C with an object o_1 is not explicitly represented by one property in the factorized RDF graph. In order to retrieve o_1 , a path across the labeled edges between entities s

and the corresponding surrogate entity sg have to be traversed. Similarly, the rest of the transformation rules establish how explicit associations between entities of C and the objects using properties p_2, \dots, p_n in the original RDF graphs are represented by the path of labeled edges annotated with properties in the factorized RDF graphs. Algorithm 3 adds the corresponding labeled edges of these paths in lines 7-16. Furthermore, Figure 6b presents a portion of the RDF graph in Figure 1a and corresponding transformation in the factorized RDF graph in Figure 3c.

Algorithm 3 The Factorization Algorithm

Input: An RDF graph $G(V, E, L)$, A class C , A set SP of properties from $E.FSP$ Algorithm 1 or $G.FSP$ Algorithm 2

Output: Factorized RDF Graph $G'(V', E', L')$ and entity mappings μ_N

```
1:  $\mu_N \leftarrow \emptyset, V' \leftarrow \emptyset, E' \leftarrow \emptyset, L' \leftarrow \emptyset$ 
2: for all  $o_1, o_2, \dots, o_n \in V$  such that  $SS = \{s|p_1, p_2, \dots, p_n \in SP \text{ AND } (s \text{ type } C) \in G, (s \text{ } p_1 \text{ } o_1) \in G, (s \text{ } p_2 \text{ } o_2) \in G, \dots, (s \text{ } p_n \text{ } o_n) \in G\}$  do
3:    $sg \leftarrow \text{SurrogateEntity}()$ 
4:   for  $ss \in SS$  do
5:      $\mu_N \leftarrow \mu_N \cup \{(ss, sg)\}$ 
6:   end for
7: end for
8: for  $(s \text{ } p \text{ } o) \in E \wedge s, o \in V$  do
9:   if  $\mu_N(s) \neq \emptyset$  then
10:    {Create compact RDF molecule}
11:    if  $p == \text{type}$  then
12:       $E' \leftarrow E' \cup \{(s \text{ } \text{instanceOf} \text{ } \mu_N(s)), (\mu_N(s) \text{ } p \text{ } o)\}$ 
13:       $V' \leftarrow V' \cup \{s, \mu_N(s), o\}$ 
14:       $L' \leftarrow L' \cup \{p, \text{instanceOf}\}$ 
15:    else if  $p \in SP$  then
16:       $E' \leftarrow E' \cup \{(\mu_N(s) \text{ } p \text{ } o)\}$ 
17:       $V' \leftarrow V' \cup \{\mu_N(s), o\}$ 
18:       $L' \leftarrow L' \cup \{p\}$ 
19:    else
20:       $E' \leftarrow E' \cup \{(s \text{ } p \text{ } o)\}$ 
21:       $V' \leftarrow V' \cup \{s, o\}$ 
22:       $L' \leftarrow L' \cup \{p\}$ 
23:    end if
24:  else
25:     $E' \leftarrow E' \cup \{(s \text{ } p \text{ } o)\}$ 
26:     $V' \leftarrow V' \cup \{s, o\}$ 
27:     $L' \leftarrow L' \cup \{p\}$ 
28:  end if
29: end for
30: return  $G'(V', E', L'), \mu_N$ 
```

The surrogate entity and the new labeled edges are highlighted in bold in the factorized RDF graph. The Algorithm 3 creates the surrogate entity in line 4; new edges are created in line 10. Additionally, assumptions about the characteristics of the entity associations in the graph are presented. Some edges existing between the entities in RDF graph in Figure 1a are not present in the factorized RDF graph in Figure 3c, these entity associations can be obtained by traversing the factorized RDF graph as indicated by the corresponding transformation rules in Figure 6a.

Figure 7 illustrates an example of factorization overhead, i.e., a case when it is not worthy to factorize a class given a set of properties in an RDF graph. Figure 7a presents an example where savings are observed in the number of edges after factorization. The factorization of RDF graph in Figure 7a for the class C using the properties p_1 , p_2 , and p_3 , reduces the number of edges from 20.0 to 12.0 and the positive value 40.0% for percentage savings indicates a percentage decrease in the number of edges. Furthermore, the edge savings gained after factorization are high enough to compensate the addition of the surrogate entity cM in the factorized RDF graph. In contrast, factorization of the RDF graph over the class C using the properties p_1 and p_2 introduces an overhead, as shown in Figure 7b, by increasing the number of nodes and edges in the factorized RDF graph. The number of edges is increased from 18.0 to 22.0, shown in Figure 7b, after factorization and a negative value -22.0% for the percentage savings indicates an increase in the number of edges. The star patterns, detected in the original RDF graph, in Figure 7b, are replaced by the corresponding compact RDF molecules with the corresponding surrogate entity and new labeled edges (presented in Algorithm 3). Due to the high number of star patterns, the addition of the surrogate entities and new labeled edges increases the size of the factorized RDF graph.

5 Experimental Study

We study the effectiveness of the proposed techniques for detecting frequent star patterns. Moreover, given a class, we evaluate the impact of the factorization

techniques over the RDF graphs size by selecting several combinations of the properties in the class. We empirically assessed the following research questions: **RQ1)** Are the proposed frequent star patterns detection techniques able to efficiently detect the frequent star patterns in RDF graphs? **RQ2)** Are the proposed frequent star patterns detection techniques able to detect the frequent star patterns in RDF graphs? **RQ3)**

What is the impact of different combinations of properties of a class over the size of factorized RDF graphs? **RQ4)** Are the proposed factorization techniques able to reduce the number of labeled edges in RDF graphs? Our experimental configuration is as follows:

Datasets. Evaluation is conducted on three *Linked-SensorData* datasets [22] semantically described using the Semantic Sensor Network (SSN) Ontology. These RDF datasets comprise observations and measurements of several climate phenomena, e.g., temperature, visibility, precipitation, rainfall, and humidity, collected during the hurricane and blizzard seasons in the United States in the years 2003, 2004, and 2005³. Table 1a describes the main characteris-

tics of these RDF datasets. Moreover, Figure 8 shows percentage of repeated RDF triples with wind speed, temperature, and relative humidity values in dataset *D1D2D3*. The unit of measurement is same for each type of observation. These plots show that some of the large number of observed values are highly repeated in the datasets. Further, values are not discretized to produce the same query answers.

Metrics. We measure the results of our empirical evaluation in terms of number of nodes and edges in an RDF graph. The size of an RDF graph is presented as the sum of nodes and edges in the graph, where the nodes correspond to the entities and objects, whereas the edges are labeled edges annotated with the properties of a class in an RDF graph. In our empirical evaluation, we report on the following metrics: **a) Execution Time (Exec.Time(ms))** is the time required to find the frequent star patterns in RDF graphs. **b) Number of Nodes (NN)** is the number of *Observation* and *Measurement* entities and objects in RDF graphs. **c) Number of Labeled Edges (NLE)** represents the number of labeled edges annotated with the properties in *Observation* and *Measurement* classes in RDF graphs. **d) Percentage Savings in the Number of Labeled Edges (%Savings)** stands for the percentage

³Available at: <http://wiki.knoesis.org/index.php/LinkedSensorData>

Table 1: **Datasets.** (a) Statistics of the datasets with observations about several weather phenomena, collected from around 20,000 weather stations in the United States; (b) The number of labeled edges $NLE(G)$, in the datasets obtained after gradually integrating the RDF datasets D1, D2, and D3 describing observations.

(a) Statistics of datasets collected from around 20,000 weather stations in the US.

Dataset ID	Climate Event	Date	#RDF Triples	# Obs
D1	Blizzard	April, 2003	38,054,493	4,092,492
D2	Hurricane Charley	August, 2004	108,644,568	11,648,607
D3	Hurricane Katrina	August, 2005	179,128,407	19,233,458

(b) Number of Labeled Edges $NLE(G)$ in datasets.

Dataset ID	Observation $NLE(G)$	Measurement $NLE(G)$
D1	24,142,314	12,071,157
D1D2	93,286,824	46,643,412
D1D2D3	207,630,306	103,815,153

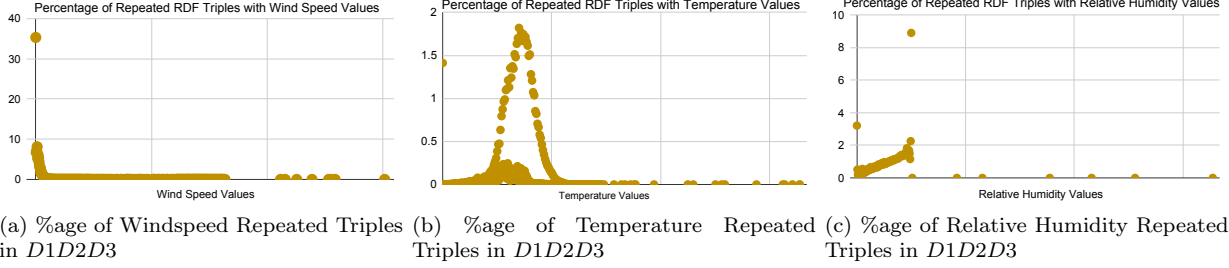


Figure 8: **Percentage of Repeated RDF Triples with Observation Values.** Few of the large number of values are highly repeated. (a) Percentage of repeated RDF triples with windspeed values; (b) Percentage of repeated triples with temperature values; (c) Percentage of repeated triples with relative humidity values.

increase or decrease in the number of labeled edges using a positive or a negative value, respectively. The interpretation of the metric **%Savings** is, higher is better.

Implementation. The experiments were performed on a Linux Debian 8 machine with a CPU Intel Xeon(R) Platinum 8160 2.10GHz and 754GB RAM. The datasets are factorized for *Observation* and *Measurement* classes using all possible combinations of the properties in each class. Table 2 shows the set of properties for *Observation* and *Measurement* (Meas.), respectively, in the SSN ontol-

ogy. Each set of properties is assigned a set identification string *SID*, and are referred with the corresponding identification string in the paper. *Observation* contains *property*, *procedure*, *generatedBy*, and *time* property. *procedure* and *generatedBy* are symmetric properties and are considered together in the sets. Similarly, in *Measurement*, sets of properties contain the properties *value* and *unit*. Further, for experiments, datasets are gradually merged to increase datasets size. The source code is available at <https://github.com/SDM-TIB/Graph-Factorization>.

Table 2: **Observation and Measurement Classes.** Sets of properties containing different properties of the *Observation* and *Measurement* (Meas.) classes in the SSN ontology, each set of properties is assigned a unique ID, e.g., *A1* and *A8*.

Class	Set of Properties	SID
Observation	{property}	A1
	{time}	A2
	{procedure, generatedBy}	A3
	{property, procedure, generatedBy, time}	A4
	{property, procedure, generatedBy}	A5
	{property, time}	A6
	{procedure, time, generatedBy}	A7
Meas.	{value, unit}	A8
	{value}	A9
	{unit}	A10

5.1 Efficiency of Frequent Star Patterns Detection Approach

For evaluating the efficiency of the proposed frequent star patterns techniques and to answer the research question **RQ1**, we execute *E.FSP* and *G.FSP* over five percent of RDF triples from dataset *D1*. The dataset of the selected RDF triples describe the *Measurement* and *Observation* classes, where several different types of observations from the *Observation* class are included in the dataset. gSpan [30] is used to generate the frequent patterns space for *E.FSP*, which iterates over all the generated frequent patterns. To evaluate the efficiency of two approaches, we selected five percent of RDF triples from the dataset *D1*; this number was chosen as a timeout because gSpan was able to generate the frequent patterns within thirty minutes. Efficiency comparison in terms of execution time of *E.FSP* and *G.FSP* is reported in Table 3. *G.FSP* finds the frequent star patterns without generating all the star patterns involving all the possible subsets of properties. Table 3 shows for *E.FSP* and *G.FSP*, the number of iterations over sets of properties *PSIterations*, the number of frequent star patterns detected *#FSP*, and the execution time in milliseconds *Exec.Time(ms)* required to detect the frequent star patterns. The results indicate that *E.FSP* and *G.FSP* detect the same frequent star patterns. The frequent star patterns, detected by *E.FSP* and *G.FSP*, are over the set of properties *A5* and *A8* for all the different observations in the *Observation* class, and the *Measurement* class, respectively. Execution time of *G.FSP* to detect frequent star patterns is less by at least three orders of magnitude than the execution time of *E.FSP*, e.g., *G.FSP* detects frequent star patterns in measurement class in 1.9×10^2 milliseconds, whereas 5.3×10^5 milliseconds are required using *E.FSP*.

5.2 Effectiveness of Frequent Star Patterns Detection Approach

To answer the research questions **RQ2** and **RQ3**, we compute the values of Formula 1 for all the sets of properties given in Table 2 for the *Observation* and *Measurement* classes, respectively, in the three RDF

datasets. The computed formula values for the *Observation* and *Measurement* classes are shown in Table 4. Moreover, we compute the size of the original and factorized RDF graphs, in terms of nodes and edges in the RDF graphs. The formula values are computed for the sets of properties that contain only one property in the set, as well as the factorization is performed using these sets of properties to illustrate the association between the formula values and the savings obtained in the factorized graphs. Table 4 shows that the set *A5* of properties in the *Observation* class generates the smaller values $D1 = 4,142,727$, $D1D2 = 15,756,888$, and $D1D2D3 = 334,898,603$ for the Formula 1, than all the other sets *A1*, *A2*, *A3*, *A4*, *A6*, and *A7*. A smaller formula value for *A5* indicates that the RDF graphs encapsulate a minimum number of star patterns, over the properties in the set *A5* such that a large number of entities of the *Observation* class match these star patterns. Therefore, replacing these star patterns with the compact RDF molecules during the factorization reduces the size of the RDF graphs. Figure 9a presents the number of *Observation* nodes *NN* and the labeled edges *NLE* in the original and factorized RDF datasets *D1*, *D1D2*, and *D1D2D3*. The results show that factorization of the *Observation* class over the set *A5* of properties reduces the sum of the number of observation nodes and the labeled edges in the factorized RDF graphs by up to 37%. On contrary, a large formula value for *A4* in datasets $D1 = 4,142,727$, $D1D2 = 15,756,888$, and $D1D2D3 = 334,898,603$, than the other sets *A1*, *A2*, *A3*, *A5*, *A6*, and *A7* indicates that a large number of star patterns over the properties in *A4* exist in the RDF graphs and a small number of entities of the *Observation* class match these star patterns. Figure 9a depicts an increase in the number of *Observation* nodes *NN* and the labeled edges *NLE* in the factorized RDF datasets *D1*, *D1D2*, and *D1D2D3* after factorizing over the properties in *A4*. Similarly, the results for *A1*, *A2*, *A3*, *A6*, and *A7* in Table 4 and Figure 9a clearly show that the higher the formula value for a set of properties increases the number of nodes and edges in the factorized RDF graphs by factorizing using the properties in the corresponding set. In case of the *Measurement* class Table 4 shows smaller formula values for the set

A8 of properties i.e., $D1 = 28,491$, $D1D2 = 34,554$, and $D1D2D3 = 40,302$, than the other sets A9 and A10. Figure 9b reports the sum of the nodes and the labeled edges representing measurements in the original and factorized RDF datasets $D1$, $D1D2$, and $D1D2D3$. The sum of the nodes and the labeled edges of the measurements are reduced up to 60% in all the factorized RDF graphs by factorizing over the properties in A8. Furthermore, the higher formula values for the sets A9 and A10 indicate less savings after factorization compared to the set A8. The number of nodes and edges in the factorized RDF graphs by factorizing over the properties in sets A9 and A10 in Figure 9b are higher than A8. These results show that the different combinations of class properties impact the factorization of RDF graphs and the proposed frequent star patterns detection techniques are able to detect the set of properties involved in the generation of frequent star patterns. Moreover, our techniques are able to anticipate the best set of properties, answering thus, research questions **RQ2** and **RQ3**.

5.3 Effectiveness of RDF Graph Factorization

We factorize the gradually increasing RDF datasets $D1$, $D1D2$, and $D1D2D3$ over the *Observation* and *Measurement* classes using the properties in the sets of properties given in Table 2. The percentage savings are computed in terms of labeled edges for the observations and measurements in the RDF datasets after factorization. Table 1b presents the number of edges $NLE(G)$ in the *Observation* and *Measurement* classes in the original RDF datasets $D1$, $D1D2$, and $D1D2D3$. Table 5 presents the number of labeled edges $NLE(G')$ and the percentage savings $\%savings$ after factorization of the *Observation* and *Measurement* classes. The highest savings 49.14%, 49.43%, and 49.53% in $NLE(G')$ for observations after factorizing $D1$, $D1D2$, and $D1D2D3$ over the properties in A5, shows that the number of frequent star patterns over the properties in A5 are reduced by replacing them with the corresponding compact RDF molecules. On the other hand, the set A4 of properties gives negative values of percentage savings

Table 3: **Efficiency of Frequent Star Patterns Detection.** *E.FSP* and *G.FSP* are used to detect the frequent star patterns for the *Observation* and *Measurement* classes in the five percent of RDF triples from the dataset $D1$. *E.FSP* and *G.FSP* detect the same frequent star patterns involving the sets A5 and A8 of properties from the *Observation* and *Measurement* classes, respectively. *G.FSP* takes less time to identify the same frequent star patterns than the time taken by *E.FSP*.

		PSIterations		#FSP		Exec.Time(ms)	
Class		E.FSP	G.FSP	E.FSP	G.FSP	E.FSP	G.FSP
Observation	Precipitation	8	4	23	23	2.1×10^4	1.5×10^1
	Pressure	5	4	183	183	1.3×10^6	7.1×10^2
	Rainfall	5	4	533	533	1.3×10^6	8.0×10^2
	RelativeHumidity	5	4	341	341	1.3×10^6	7.5×10^2
	Snowfall	8	4	382	382	9.2×10^5	3.1×10^2
	Temperature	5	4	395	395	1.3×10^6	7.8×10^2
	Visibility	5	4	395	395	1.3×10^6	7.3×10^2
	WindDirection	5	4	350	350	1.3×10^6	7.5×10^2
	WindSpeed	5	4	410	410	1.3×10^6	7.6×10^2
Measurement		1	1	1,907	1,907	5.3×10^5	1.9×10^2

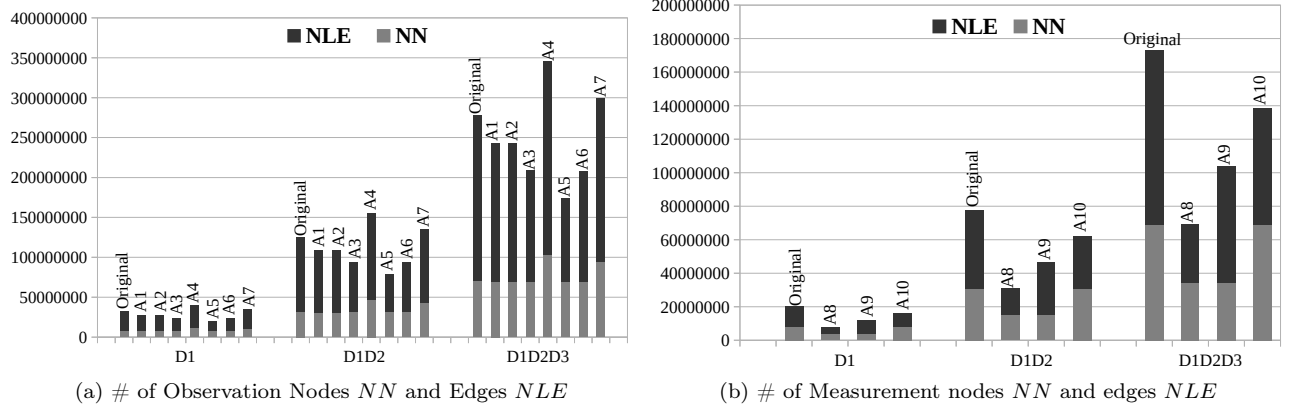


Figure 9: **Nodes and Labeled Edges.** The number of nodes NN and labeled edges NLE before and after factorization of the RDF datasets. (a) The number of nodes NN and labeled edges NLE representing observations in the RDF datasets; (b) The number of nodes NN and labeled edges NLE representing measurements.

Table 4: **Values Computed for Formula 1.** The sets of properties in Table 2 for the *Observation* and *Measurement* (Meas.) classes, respectively, are used to compute the Formula 1 values over the RDF datasets $D1$, $D1D2$, and $D1D2D3$. The minimum formula values for the *Observation* and *Measurement* classes and the corresponding sets $A5$ and $A8$, respectively, of properties are highlighted in bold. The smaller formula values for $A5$ and $A8$ in the *Observation* and *Measurement* classes, respectively, indicate the maximum savings after factorizing the RDF graphs over the properties in $A5$ and $A8$, as shown in Figure 9 and Table 5.

		#Edges(SP, C, G)		
SID		D1	D1D2	D1D2D3
Observation	A1	12,071,185	46,643,440	103,815,183
	A2	12,090,195	46,687,690	103,891,717
	A3	8,111,623	31,205,888	69,358,875
	A4	20,118,595	78,698,580	174,865,870
	A5	4,142,727	15,756,888	34,898,603
	A6	8,097,964	31,245,605	69,474,786
	A7	15,784,707	61,406,644	135,902,747
Meas.	A8	28,491	34,554	40,302
	A9	4,037,067	15,563,838	34,623,579
	A10	4,023,731	15,547,816	34,605,063

$\%Savings$, -16.68% , for the RDF dataset $D1$, and -16.67% , for the RDF datasets $D1D2$ and $D1D2D3$, indicating an increase in the number of labeled edges after the factorization of the RDF datasets. Similarly, for measurements, the positive values 66.37% of percentage savings after factorizing $D1$, and 66.56% for $D1D2$ and $D1D2D3$ over $A8$ indicate a decrease in the number of labeled edges after factorization. Furthermore, the percentage savings in the set $A8$ of properties are higher than in $A9$ and $A10$. These results allow us to positively answer research question **RQ4**.

6 Conclusions and Future Work

This article presents computational methods to identify frequent star patterns and to generate a *factorized RDF graph*, with a minimized number of frequent star patterns. A frequent star pattern contains class entities linked to the objects or other resources using labeled edges annotated with properties in the class. These frequent star patterns introduce redundancy in terms of edges and nodes. Our proposed computational methods implement the frequent star pattern detection algorithm based on search space

pruning techniques to identify the classes and properties involved in frequent star patterns. Furthermore, the proposed factorization techniques generate compact representation of RDF graphs, *factorized RDF graph*, by replacing a frequent star pattern with a compact RDF molecule, composed of a surrogate entity connected to the object in the frequent star pattern using the labeled edges annotated with relevant properties. We empirically study the effectiveness of the frequent star pattern detection algorithm to identify class and properties involved in the frequent star pattern. Furthermore, we evaluate the impact of the factorization techniques over the gradually increasing RDF graphs size and different combinations of class properties. Experimental results suggest that the proposed computational methods successfully identify the class properties involved in the frequent star patterns and remove redundancy caused by these frequent star patterns. For the best set of properties, identified by the frequent star pattern detection algorithm, the RDF graph size is reduced by up to 66.56% . Our work broadens the repertoire of techniques for representing and storing knowledge graphs by providing RDF graph compression techniques which exploit the semantics encoded in the data; these techniques generate compact rep-

Table 5: **Percentage Savings in Labeled Edges after Factorization.** Savings $\%Savings$ in the number of Labeled Edges $NLE(G')$ after factorization of the RDF datasets using the sets of properties in Observation and Measurement classes.

	SID	D1		D1D2		D1D2D3	
		NLE(G')	$\%Savings$	NLE(G')	$\%Savings$	NLE(G')	$\%Savings$
Observation	A1	20,125,493	16.64	77,745,918	16.66	173,032,155	16.66
	A2	20,144,503	16.56	77,790,168	16.61	173,108,689	16.63
	A3	16,226,021	32.79	62,546,938	32.95	139,064,503	33.02
	A4	28,170,155	-16.68	108,838,750	-16.67	242,239,479	-16.67
	A5	12,277,576	49.14	47,175,356	49.43	104,786,128	49.53
	A6	16,150,898	33.10	62,317,489	33.20	138,639,234	33.23
	A7	23,837,352	1.26	92,088,523	1.28	204,304,156	1.60
Meas.	A8	4,059,738	66.37	15,599,469	66.56	34,716,176	66.56
	A9	8,069,688	33.15	31,130,127	33.26	69,300,827	33.25
	A10	8,056,352	33.26	31,114,105	33.29	69,282,311	33.26

representations of RDF graphs to help improving query processing over RDF graphs without requiring a customized engine. Our work contributes to the crucial knowledge graph representation and provides the basics for further development of the efficient processing techniques over the compact knowledge graphs. In future, we will exploit parallel processing to efficiently find frequent star patterns.

Acknowledgments

Farah Karim is supported by the German Academic Exchange Service (DAAD); this work is partially funded by the EU H2020 project IASiS (GA No.727658).

References

- [1] D. Abadi, S. Madden, and M. Ferreira. Integrating compression and execution in column-oriented database systems. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 671–682. ACM, 2006.
- [2] D. Allen, A. Hodler, M. Hunger, M. Knobloch, W. Lyon, M. Needham, and H. Voigt. Understanding trolls with efficient analytics of large graphs in neo4j. *BTW 2019*, 2019.
- [3] S. Álvarez-García, N. R. Brisaboa, J. D. Fernández, and M. A. Martínez-Prieto. Compressed k2-triples for full-in-memory rdf engines. *arXiv preprint arXiv:1105.4004*, 2011.
- [4] M. Arenas, C. Gutierrez, and J. Pérez. Foundations of RDF databases. In *Reasoning Web. Semantic Technologies for Information Systems*, pages 158–204. Springer, 2009.
- [5] S. Auer, V. Koltun, M. Prinz, A. Kasprzik, M. Stocker, and M. Vidal. Towards a knowledge graph for science. In *Proceedings of the 8th International Conference on Web Intelligence, Mining and Semantics, WIMS 2018*, 2018.
- [6] C. Bizer, T. Heath, and T. Berners-Lee. Linked data: The story so far. In *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227. IGI Global, 2011.
- [7] P. A. Boncz, M. Zukowski, and N. Nes. Monetdb/x100: Hyper-pipelining query execution. In *Cidr*, volume 5, pages 225–237, 2005.
- [8] N. R. Brisaboa, S. Ladra, and G. Navarro. k2-trees for compact web graph representation. In *International Symposium on String Processing and Information Retrieval*, pages 18–30. Springer, 2009.
- [9] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, et al. 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, 2012.
- [10] G. P. Copeland and S. N. Khoshafian. A decomposition storage model. In *Acm Sigmod Record*, volume 14, pages 268–279. ACM, 1985.
- [11] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis. Grami: Frequent subgraph and pattern mining in a single large graph. *Proceedings of the VLDB Endowment*, 7(7):517–528, 2014.
- [12] P. Ernst, A. Siu, and G. Weikum. Knowlife: a versatile approach for constructing a large knowledge graph for biomedical sciences. *BMC bioinformatics*, 16(1):157, 2015.
- [13] J. D. Fernández, A. Llaves, and Ó. Corcho. Efficient RDF interchange (ERI) format for RDF data streams. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II*, pages 244–259, 2014.
- [14] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias. Binary rdf representation for publication and exchange

- (hdt). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19:22–41, 2013.
- [15] I. Grangel-González, L. Halilaj, M. Vidal, O. Rana, S. Lohmann, S. Auer, and A. W. Müller. Knowledge graphs for semantically integrating cyber-physical systems. In *Database and Expert Systems Applications - 29th International Conference*, 2018.
 - [16] A. K. Joshi, P. Hitzler, and G. Dong. Logical linked data compression. In *Extended Semantic Web Conference*, pages 170–184. Springer, 2013.
 - [17] F. Karim, M. N. Mami, M.-E. Vidal, and S. Auer. 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, 2017.
 - [18] O. Lassila, R. R. Swick, et al. Resource description framework (rdf) model and syntax specification. 1998.
 - [19] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al. Dbpedia—a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.
 - [20] M. Meier. Towards rule-based minimization of rdf graphs under constraints. In *International Conference on Web Reasoning and Rule Systems*, pages 89–103. Springer, 2008.
 - [21] J. Z. Pan, J. M. G. Pérez, Y. Ren, H. Wu, H. Wang, and M. Zhu. Graph pattern based rdf data compression. In *Joint International Semantic Technology Conference*, pages 239–256. Springer, 2014.
 - [22] H. K. Patni, C. A. Henson, and A. P. Sheth. Linked sensor data. 2010.
 - [23] R. Pichler, A. Polleres, S. Skritek, and S. Woltran. 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, 2010.
 - [24] E. Prud’hommeaux and A. Seaborne. Sparql query language for rdf. w3c recommendation (january 15, 2008), 2011.
 - [25] M. A. Roth and S. J. Van Horn. Database compression. *ACM Sigmod Record*, 22(3):31–39, 1993.
 - [26] A. Singhal. Introducing the knowledge graph: things, not strings. *Official google blog*, 5, 2012.
 - [27] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O’Neil, et al. C-store: a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, pages 553–564. VLDB Endowment, 2005.
 - [28] M.-E. Vidal, K. M. Endris, S. Jazashoori, A. Sakor, and A. Rivas. Transforming heterogeneous data into knowledge for personalized treatments a use case. *Datenbank-Spektrum*, pages 1–12.
 - [29] T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte. The implementation and performance of compressed databases. *ACM Sigmod Record*, 29(3):55–67, 2000.
 - [30] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724. IEEE, 2002.
 - [31] M. Zhu, W. Wu, J. Z. Pan, J. Han, P. Huang, and Q. Liu. Predicate invention based rdf data compression. In *Joint International Semantic Technology Conference*, pages 153–161. Springer, 2018.
 - [32] M. Zukowski, S. Heman, N. Nes, and P. A. Boncz. Super-scalar ram-cpu cache compression. In *Icde*, volume 6, page 59, 2006.