Computational Methods and Optimizations for Containment and Complementarity in Web Data Cubes

Marios Meimaris

ATHENA Research Center

George Papastefanatos

ATHENA Research Center

Panos Vassiliadis

University of Ioannina

Ioannis Anagnostopoulos

University of Thessaly

Abstract

The increasing availability of diverse multidimensional data on the web has led to the creation and adoption of common vocabularies and practices that facilitate sharing, aggregating and reusing data from remote origins. One prominent example in the Web of Data is the RDF Data Cube vocabulary, which has recently attracted great attention from the industrial, government and academic sectors as the de facto representational model for publishing open multidimensional data. As a result, different datasets share terms from common code lists and hierarchies, this way creating an implicit relatedness between independent sources. Identifying and analyzing relationships between disparate data sources is a major prerequisite for enabling traditional business analytics at the web scale. However, discovery of instance-level relationships between datasets becomes a computationally costly procedure, as typically all pairs of records must be compared. In this paper, we define three types of relationships between multidimensional observations, namely *full containment*, *partial containment* and *complementarity*, and we propose four methods for efficient and scalable computation of these relationships. We conduct an extensive experimental evaluation over both real and synthetic datasets, comparing with traditional query-based and inference-based alternatives, and we show how our methods provide efficient and scalable solutions.

Keywords: `elsarticle.cls`, LaTeX, Elsevier, template
2010 MSC: 00-01, 99-00

1. Introduction

The increasing adoption of RDF as the de facto Semantic Web standard has led the industrial, government, and academic sectors to leverage Linked Data technologies [14, 58] in order to publish, re-use and extend big amounts of proprietary data. A large subset of data on the web consists of multidimensional data about policies, demographics, socio-economics and health data among others [53].

Statistical multidimensional data is often represented in the form of data cubes. Under this model, a single data record, named *observation* or *fact*, is broadly defined as the value of a specific measure over several different observed dimensions [13]. For example, Germany's population for the year 2001 can be represented as an observation with *population* as the measure, and *location* and *time* as the dimensions, with the values *Germany* and *2001* instantiating these dimensions. The use of hierarchical values enables the representation of information on multiple combinations of levels, such as the *female population of a country in the last decade*, or *the total population of a city in the last year*. An example of dimension hierarchies can be seen in Figure 1. The RDF Data Cube Vocabulary (QB) [14] provides a schema for RDF multidimensional data, allowing for the representation of schemas, dimensions, measures, hierarchies, observations, among others. Considering the aforementioned example, its mapping to the RDF QB vocabulary can be seen in Listing 1. RDF QB enables different data publishers to fit their data in a common meta-schema, and reuse common entities across different sources. Hence, remote datasets often exhibit overlaps in the values that instantiate their dimensions and measures, this way creating implicit relationships between observations among remote sources; for example, an observation can be a specialization or generalization of another observation from a different dataset, an observation can partially aggregate information contained in other observations, or finally different observations can capture complementary knowledge and can be combined together.

In this article, inspired by the notion of fusion cubes [1] towards self-service analytics, we define instance-level relationships for multidimensional observations, and we address the challenge of efficient computation of these relationships over multiple data cubes.

In order to better illustrate the defined relationships, we discuss an example scenario that will be used throughout the paper. In this scenario, the user has gathered data from several remote sources in order to explore unemployment and population demographics. The gathered data are in the form of observations and originate from Linked Data sources. As such, they exhibit heavy re-use of the same hierarchies and code lists[1]. The example hierarchies are shown in Figure 1, and a snapshot of the gathered data is shown in Figure 2, where the

[1]Some degree of schema alignment is often necessary in realistic scenarios. This type of alignment is used in the following two prominent cases: (a) traditional BI settings, where all dimensions provide a reconciled dimension bus, and (b) user-initiated data collections from the web.

analyst has gathered data from three different datasets, namely D_1, D_2 and D_3.

Observations o_{11} and o_{31} have the same values for the *refArea* and *refPeriod* dimensions, while the *sex* dimension has the most general value possible, i.e., *Total*. Intuitively, this means that the two observations measure different things for the same setting, and are thus *complementary*. Furthermore, observations o_{21}, o_{22} measuring unemployment in Greece and Italy for the year 2001, are generalizations of o_{32} and o_{33}, because the latter measure unemployment in Athens and Rome, which are sub-parts of Greece and Italy respectively, for a sub-period of 2011. For the data of the example, these relationships can be seen in Figure 3.

Discovering relationships such as the above is useful in several tasks. Multidimensional data enable third parties to study, process and visualize information in order to perform more complex analytics such as combining different datasets, discovering new knowledge, assisting socio-centric processes such as data journalism, as well as enabling evidence-based policy making on the government and industry levels[41, 9, 47]. As potential users, we consider data scientists, such as data analysts or, data journalists and business users, who collect data from external and corporate sources in their personal data cube for analysis purposes. Then, the added value of detecting such relationships can be summarized in the following. First and foremost, observations that originate from different datasets become linked, and thus comparable in future analytical tasks, as in the case of fusion cubes [1] towards self-service analytics. Furthermore, navigation and exploration of aggregations of datasets is facilitated with the existence of links on the instance level. Traditional OLAP tasks such as rolling up or drilling down can be applied for the exploration of remote cubes. These types of relationships can help quantify the degree of relatedness across remote datasets and this way provide recommendations for online browsing. Finally, materializing these relationships speeds up online exploration, as well as computation of k-dominance [12], skylines and k-dominant skylines.

Finding implicit knowledge across different sources is a non-trivial, computationally challenging task [9], that is inherently quadratic at its core, since all pairs of records must be examined. Traditional query processing methods such as SPARQL engines, and inference-based methods fail to address this issue efficiently as the volume of data increases. For instance, our experiments with recursive, property-path based SPARQL queries show that even for small numbers of records (20,000 observations from 7 datasets) require more than one hour in commodity hardware to detect and materialize pair-wise containment relationships. Similary, inference-based methods such as SWRL [28] and Jena Rules [10] fail to scale due to the transitive nature and the universal restrictions of these relationships; the search space expands exponentially [17]. Hence, the need arises to establish more efficient methods that can scale to the size of the web of data.

Approach Overview. In this paper, we address efficient computation of three specific types of relationships in multidimensional data from different sources, namely *full containment*, *partial containment* and *complementarity*, by extending the work presented in [37][38]. Full containment between observations

occurs when all dimension values in two observations are hierarchically related in the same direction, i.e., the containing observation is a generalization of the contained observation, while partial containment occurs when at least one, but not all of the dimension values are hierarchically related. Complementarity occurs when two observations identify the same setting but measure different aspects, and thus hold complementary information. Specifically for complementarity, we extend the notion of *schema complement*[15] to fit observations.

We present a quadratic baseline algorithm for computation of these relationships, and introduce three alternative methods that target efficiency and scalability, an approach based on pruning the required comparisons by clustering together related observations, and two approaches based on the notion of the multidimensional lattice, a data structure that groups observations based on their defined combination of dimension levels. The first approach exploits the dimension levels in order to reduce the required comparisons, whereas the optimized approach makes use of the inherent hierarchical structure of the lattice to further speed up the detection process. We perform an extensive experimental evaluation of the 4 methods over 7 real-world multidimensional datasets, and compare their efficiency with two traditional approaches, namely SPARQL querying and rule-based inferencing. Finally, we evaluate the scalability of our approach in an artificially generated dataset.

A first attempt to defining and computing these relationships was presented in [38]. This work informally introduced these relationships and proposed the naive and the 2 alternative methods for calculating them. This paper extends the work presented in [38] by providing formal definitions for the problem components, defining a new optimized method that drastically outperforms previous methods, and extending the experimental evaluation to assess the defined optimization with respect to the previous methods.

Contributions. The contributions of this paper are summarized as follows:

- we formally define the notions of *full containment*, *partial containment* and *complementarity*, originally introduced in [38],
- we present four algorithms, a baseline, data-driven technique for computing these properties in memory, and three alternative approaches with the scope of improving performance with respect to efficiency and scalability,
- we perform an extensive experimental evaluation of the achieved efficiency and scalability over both real-world and synthetic datasets, comparing between the proposed methods, a SPARQL-based and a rule-based approach.

The remainder of this paper is organized as follows. Section 2 discusses related work, Section 3 presents the preliminary definitions and formulates the problem. Section 4 presents the proposed approaches. Section 5 describes the experimental evaluation. Finally, Section 6 concludes this paper.

```
ex:obs1 a qb:Observation ;
  qb:dataSet ex:dataset ;
  ex:refPeriod ex:Y2001 ;
  sdmx-attr:unitMeasure ex:unit ;
  ex:refArea ex:DE ;
  ex:population "82,350,000"^^xmls:integer .
```

Listing 1: Example RDF Data Cube observation

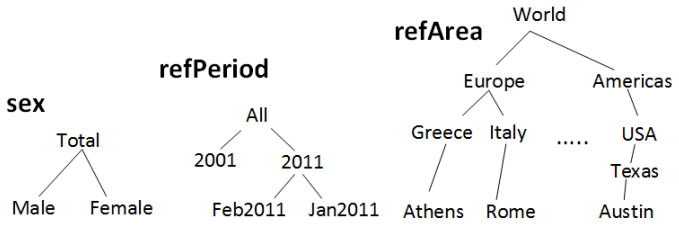


Figure 1: Hierarchical code list for the dimensions in Figure 2.

D_1

	refArea	refPeriod	sex	ex:Population
o_{11}	Athens	2001	Total	5M
o_{12}	Austin	2011	Male	445K
o_{13}	Austin	2011	Total	885K

D_2

	refArea	refPeriod	ex:Unemployment	ex:Poverty
o_{21}	Greece	2011	26%	15%
o_{22}	Italy	2011	20%	10%

D_3

	refArea	refPeriod	ex:Unemployment
o_{31}	Athens	2001	10%
o_{32}	Athens	Jan2011	30%
o_{33}	Rome	Feb2011	7%
o_{34}	Ioannina	Jan2011	15%
o_{35}	Austin	2011	3%

Figure 2: Candidate relationships between observations.

	refArea	refPeriod	sex	ex:Unemployment	ex:Poverty	ex:Population
o_{21}	Greece	2011	Total	26%	15%	-
contains:	***contains:***	***contains:***				
--o_{32}	*--Athens*	*--Jan2011*	*Total*	*30%*	-	-
--o_{34}	*--Ioannina*	*--Jan2011*	*Total*	*15%*	-	-
o_{22}	Italy	2011	Total	20%	10%	-
contains:	***contains:***	***contains:***				
--o_{33}	*--Rome*	*--Feb2011*	*Total*	*7%*	-	-
o_{11}	Athens	2001	Total	-	-	5M
complements						
--o_{31}	*Athens*	*2001*	*Total*	*10%*	-	-
o_{13}	Austin	2011	Total		-	885K
complements						
--o_{35}	*Austin*	*2011*	*Total*	*3%*		

Figure 3: Derived containment and complementarity relationships from datasets D_1, D_2 and D_3 of Figure 2.

2. Related Work

In the context of RDF, there exists a growing body of research focused on the provision of tools, methods and techniques for representing, analysing and processing multidimensional data. In this paper, we build on, and extend the work presented in [38][37], where we introduced the notions of full containment, partial containment and complementarity, and discussed three approaches for efficient computation of these relationships. We formally define the problem components and relationships, we improve upon the presented approaches, and we introduce a novel optimization of the cube masking approach that targets performance, complementing it with an extended experimental evaluation.

The general problem of detecting similarities between resources is central in the fields of entity resolution, record linkage and interlinking [39, 40, 59, 43, 19, 45]. However, these approaches are focused on finding links between resources from different datasets without taking into account multidimensional features such as dimension values. To the best of our knowledge, this is the first work centred on the definition, representation and computation of relationships between instance-level multidimensional data.

2.1. Schema-Level Hierarchy Extraction for OLAP

Traditional OLAP and data warehousing systems and frameworks are often used for performing analytical queries with operators that can generalize (roll-up) or specialize (drill-down) specific records, based on their defined dimension hierarchies. These are most commonly built upon a relational backbone, or native data cube implementations [55], and rely on the management of data from trusted sources with known schemas and interconnections. The latter assumption does not apply to data analysis in the Data Web, because the input potentially originates from remote, implicitly related sources. Furthermore, these types of systems are not built for detecting instance-level relationships such as containment and complementarity. For instance, early approaches on automatic concept hierarchy detection have been proposed, but deal with hierarchy construction on the schema or the attribute level, rather than the instance level[25][26]. Similarly, extraction of concept hierarchies from web tables and transformation to data cubes has been studied in [5], and extraction of dimension hierarchies from ontological data has been addressed in [50]. Thus, the process of detecting instance-level relationships must be translated to queries over the employed format (e.g. SQL or SPARQL queries) in the form of query operators, which makes the detection costly, as will be shown in the experiments, or derived with the use of customized ETL processes.

2.2. Analytical Mining in the presence of hierarchies

The problem of finding related observations in multidimensional data spaces has been addressed in the field of Online Analytical Mining (OLAM)[35], which refers to the integration of data mining techniques into traditional OLAP. These

methods have been successfully used for tasks such as classification of observations and detection of outliers [35][3], exploration recommendation[4][6], intelligent exploratory query recommendation [51], discovery of implicit knowledge[22] and optimized OLAP querying[35]. In [49], the authors introduce the *shrink* operator that exploits hierarchies as a means to provide summarized and shortened cubes. It achieves this by clustering together and consequently merging similar facts, in order to assist human-readability. Their work is not driven by efficiency, as in our case, rather they focus on improving the presentation of cubes in graphical form. In [11], the authors tackle the problem of performing roll-up and drill-down operations on continuous dimensions, rather than fixed dimension values as in our case, and to this end they employ hierarchical clustering on the numerical values of the dimensions.

2.3. Partial Materialization

In [18], the authors target efficiency in the performance of OLAP related tasks by studying partial materialization techniques for aggregation and summarization of multidimensional observations. Similarly, in [24] the authors propose materialized views for efficient processing of aggregation queries. These two approaches resemble our notion of observation containment, and can indeed be complemented by the efficient computation of this type of relationship on the observation level. In [60], the authors propose a probabilistic approach for providing full and partial materialization over aggregate analytics at the cube level. Ibragimov et al. [30] use materialized views formulated as SPARQL queries in order to address the lack of support for incomplete data with implicit information, and they evaluate their approach on multidimensional RDF data represented with the QB4OLAP model [20], which is an extension of the RDF QB vocabulary. This way, they provide scalable support for aggregate queries that include roll-up and drill-down exploration over incomplete data. This work is complementary to our methods for efficient computation of aggregate relationships between observations (i.e., observation containment) and the (partial) materialization of RDF views can be complemented by the optimizations presented in this article.

2.4. Skyline Computation

The computation of containment relationships has been addressed in different contexts, with *skyline computation* being the most prominent one. Specifically, skyline computation is based on the definition of observation *dominance*, and asserts the existence of points in the multidimensional dataset that are not dominated (i.e., fully contained) by other points [61, 54, 34]. The set of these points comprises the skyline of a dataset, and has found important applications in summarization and recommendation tasks in data warehousing. In this regard, full containment is a generalization of the skyline problem, where we are interested in all intermittent skylines at all of the level combinations of the hierarchies. Similarly, partial containment in the same context is referred as the *k-dominance* problem in [12], where the authors propose a methodology for

efficient computation of partial skylines in subsets of the original dimension set of the input. The problem of Subspace Skyline computation is presented in [48], and is defined as the computation of partial skylines in subsets of the dimensions of a given dataset. This is relevant to our definition of partial containment, however, partial containment can be defined in several different subsets of the dimensions between two observations at the same time, which makes our problem more computationally complicated.

2.5. Observation Relationships via Similarity Metrics

As a metric of relatedness, containment and complementarity relationships have the potential to highlight similarity between observations as well as datasets, even though this is not the main focus of this work. In this regard, Aligon et al.[4] use query features in OLAP sessions in order to define distance functions that capture instance-level similarities. In a related context, Baikousi et al.[6] propose several similarity metrics in the form of distance functions that specifically address distances in hierarchical code lists. In [29] the authors propose a set of scalable multidimensional methods via hierarchical clustering in order to measure similarity between reports in the same cubes. In the broader context of web-based data sources, the work in [15] defines the notions of *schema* and *entity complement*, the latter of which is the basis for our definition of observation complementarity.

Recent works in entity resolution (ER) have been shown to perform efficiently in cases when duplicate entities need to be identified based on pre-defined similarity metrics. As ER is mainly a quadratic problem, in the sense that all pair-wise comparisons are needed in order to identify duplicate or similar entries, these works usually focus on providing fast ways of partitioning the search space in smaller chunks, or blocks, and limiting the pair-wise comparisons of records within the same, or nearby blocks. Examples of these have been addressed in [8][43][44], while the reader is referred to [42] for an extensive experimental evaluation of recent schema-less and schema-aware techniques. While these approaches aim at identifying similar entries, they do not address cases where the examined attributes (i.e., dimension values) exhibit hierarchical relationships, as in our case. Furthermore, they provide approximate solutions, rather than exact ones.

2.6. Multidimensional Linked Data Related Approaches

The versatility of the RDF model has enabled the creation of several schemas, vocabularies and ontologies that are used for the representation of multidimensional data, concept hierarchies, code lists and so on, with the most prominent example being the RDF Data Cube Vocabulary (QB). Furthermore, many high-level representation models such as RDFS[2] and SKOS[3] provide conventions for

[2]https://www.w3.org/TR/rdf-schema/
[3]https://www.w3.org/2004/02/skos/

representing hierarchical dependencies, such as *rdfs:subClassOf*, and *skos:broader/skos:narrower*. In fact, in this work, we rely on *skos* concepts and hierarchical properties in order to detect and represent hierarchical dependencies between values in code lists that are shared among different datasets.

In the context of Linked Data, a thorough survey of how OLAP exploration tasks and processes are performed in the context of the Semantic Web, is given in [2]. The authors perform a classification of research works that leverage Semantic Web technologies for OLAP schema design and data provisioning according to five criteria, namely *materialization*, *transformations*, *freshness*, *structuredness*, and *extensibility*, and further analyzed these technologies with respect to *Reasoning*, *Computation* and *Expressivity*. In this regard, our work can be categorized as a *computational* approach with instance-level inferred materialization as the ultimate goal, in order to allow for constant-time access to more complex exploration tasks, such as querying implicit information. The work in [32] addresses the problem of finding related cube entities amongst different and remote sources with the use of an extended *Drill-Across* operator. The authors tackle relatedness on the level of the cube schema, and to that end they define relatedness by quantifying the difficulty of tasks such as conversion between cubes and merging of different cubes. In [31] the authors advocate the development of native engines that translate traditional OLAP to SPARQL queries and materialized views in order to tackle the lack of support for analytical workflows in traditional RDF management systems. In [33] the authors propose a SPARQL-based ETL framework for extracting multidimensional star-pattern data and hierarchies from RDF and Linked Data using dynamically generated SPARQL queries, but the authors note the lack of functionality regarding information extraction in the form of aggregation functions in their approach. In [21], the authors propose *CQL*, a conceptual algebra for querying multidimensional RDF data, which they use to translate SPARQL queries and apply traditional SPARQL query optimization methods. In [7], the authors propose a method for discovering and merging OLAP cubes in the context of RDF. While this is an interesting approach, it is not centred on the detection of instance-level relationships, as is the main focus in our work. In [20][57], the authors present the QB4OLAP vocabulary, an extension of the RDF QB vocabulary with OLAP constructs such dimension levels, with the aim to go beyond the representational capabilities of QB and enable native support for traditional OLAP tasks in Linked Open multidimensional datasets. Extending on this work, in [56] the authors implement a tool for performing OLAP-related tasks on QB Linked Data without requiring SPARQL expertise. To this end, they provide functionality for semi-automatic transformation of existing QB datasets to QB4OLAP, and high-level query formulation using the generic QL language. Furthermore, they implement an enrichment module that is able to extract hierarchies and code lists from remote Linked Data sources.Even though the scope of these works is not to provide efficient computation of instance-level relationships between observations, as is our focus, they are complementary to our approach.

3. Problem Definition

In this section, we present preliminaries of our approach and formulate the problem addressed in this paper. As was noted in the introduction, in the context of this work we are interested in processing linked open multidimensional datasets with OLAP cube characteristics. These datasets must exhibit several characteristics, the main of which is the conformance to a representational model that allows the description of cubes and cube facts, i.e., observations. Furthermore, linked open data technologies use commonly agreed ontologies for describing data across different sites. This enables us to process datasets which, although being published by different sources, are following the same semantics for the description of the schema, the values of the dimensions, the unit of measurements, etc. Under this scope, we consider a problem space consisting of n input datasets, each of which follows a multidimensional schema in the form of one or more cubes, containing observation instances. In the following, we present and define the components of the problem.

Definition 1. ***Dimension Schema.***

Following the definitions in [23], a dimension schema P is a tuple $(L, \rightarrow)$ where L is a non-empty finite set of values $h_1, h_2, \ldots, h_n$ along with a top value concept named All, and $\rightarrow$ is a partial ordering of the values in L. This partial ordering essentially defines a *hierarchy* in the values. In the setting of this paper, L is a fixed code list, that is represented by URIs. Furthermore, the $\rightarrow$ operator in the definition of P defines a hierarchy such that when $h_i \succ h_j$, where h_i, h_j are values in L, then h_i is a hierarhical parent of h_j. The concept $h_{root} = All$ is defined as the top level concept in each code list, i.e., an ancestor of every other value in L, such that $\forall h_i : h_{root} \succ h_i$. This hierarchical ancestry is reflexive, i.e. $\forall h_i : h_i \succ h_i$. Figure 1 shows several code list values in their respective hierarchies.

Definition 2. ***Cube Schema.***

A cube schema CS is a tuple (P, M), where P is a dimension schema, and M is a finite set of *measures*. Measures are essentially measurements of a specific metric that are instantiated over a point in the multidimensional space defined by P.

Definition 3. ***Observation.***

An observation is a cube instance that defines a single point in the multidimensional space. More specifically, an observation o is a tuple of the form $o_a = (h_a^1, h_a^2, \ldots, h_a^l, v_a^1, v_a^2, \ldots, v_a^m)$, where h_a^i is the value of dimension P_i, and v_a^i is the value of measure M_i for observation o_a. In other words, an observation is an entity that instantiates all of the dimensions and measures that are defined in its respective dataset. In our running example, the values in the white cells represent dimension values (e.g. *"Athens"* is a value for dimension *refArea*), while grey cells represent the values of measures, such as *10% unemployment*.

Definition 4. ***Dataset Structure.***

Let D = $\{D_1, \ldots, D_n\}$ be the set of all input datasets. A dataset $D_i \in D$ consists of a set of data observations $O_i = o_1, \ldots, o_k$, as well as a set of dimension schemas $\mathbf{P_i} = P_1, \ldots, P_l$ and a set of measures $\mathbf{M_i} = M_1, \ldots, M_m$. Thus, D_i is a tuple of the form (O_i, CS_i), where $CS_i = (\mathbf{P_i}, \mathbf{M_i})$. This means that D is defined as the union of the respective components of the input datasets, that is, $D = (O_D, CS_D)$, with $O_D = \bigcup_{i=1}^{n} O_i$, $CS_D = (P_D, M_D)$, with $P_D = \bigcup_{i=1}^{n} \mathbf{P_i}$ and $M_D = \bigcup_{i=1}^{n} \mathbf{M_i}$. In the running example, all three datasets D_1, D_2, D_3 share the dimensions *refArea* and *refPeriod*. Furthermore, D_2 and D_3 share the measure *ex:unemployment*.

These datasets originate from possibly remote, linked open data sources, and can exhibit overlap in both their records, and the used/reused vocabularies. Hence, dimension values that instantiate observation instances are drawn from linked open codelists and vocabularies and can be shared across datasets. This creates the possibility of linkage between datasets on the instance level, i.e., observations can be related across remote datasets. For this reason, we will define three types of relationships that pairs of observations can exhibit, namely *full containment*, *partial containment*, and *complementarity*.

Definition 5. ***Observation Complementarity.***

Complementarity is a binary relationship between a pair of observations. Specifically, we define complementarity as a function $compl : O \times O \rightarrow B$, where B is the boolean set $B = \{0, 1\}$. Let o_a and o_b be two observations that originate from datasets $D_a = (O_a, CS_a)$ and $D_b = (O_b, CS_b)$ respectively, with $CS_a = (P_a, M_a)$ and $CS_b = (P_b, M_b)$. Then, o_a complements o_b when the following conditions hold:

$$\forall P_i \in \mathbf{P_a} \cap \mathbf{P_b} : h_a^i = h_b^i \tag{1}$$

$$\forall P_j \in \mathbf{P_a} \triangle \mathbf{P_b} : h_b^j = h_{root} \tag{2}$$

where $\mathbf{P_a} \triangle \mathbf{P_b}$ is the symmetric difference of sets $\mathbf{P_a}$ and $\mathbf{P_b}$. When both conditions hold true, there is a complementarity relationship between o_a and o_b, i.e., $(1) \wedge (2) \Rightarrow compl(o_a, o_b) = 1$. We denote this with $o_a \overset{c}{=} o_b$. This definition essentially relates the two observations as occupying the same point in the multi-dimensional space defined by their shared dimensions. These shared dimensions $\mathbf{P_a} \cap \mathbf{P_b}$ must be instantiated with the same values from the respective code lists (condition (1)), and all non-shared dimensions, i.e., $\mathbf{P_a} \triangle \mathbf{P_b}$, must be equal to the root of the dimension hierarchy, i.e. the value $h_{root} = All$, thus providing no further specialization, (condition (2)). This relationship indicates that the two observations basically *identify the same setting*. The complementarity relationship is symmetric, thus $o_a \overset{c}{=} o_b$ also implies $o_b \overset{c}{=} o_a$.

For instance, in a one-dimensional setup where the only dimension is *refArea*, an observation that measures *poverty* for the value *Greece* in this dimension, exhibits complementarity with an observation that measures *population* in Greece.

If the second observation originates from a dataset that includes the dimension *sex*, then the two would complement each other only if the non-shared dimension (i.e., *sex*) provides no specialization in its respective observation. In our example, observations o_{11} and o_{31} are complementary, in that they measure different things for Athens in 2001. Condition (2) holds for o_{31} in the sex dimension, where absence of the dimension implies existence of the root value $h_{root} = All$. The fact that o_{11} refers to all values from the sex dimension does not provide any further specialization and is inherently found in o_{31} as well.

Definition 6. ***Observation Containment.***

A special type of directed relationship between a pair of observations exists when one of the two observations is a specialization of the other. We call this a *containment relationship*. For instance, the population of Greece implicitly contains all the populations of Greece's cities. However, there are cases where only a subset of the dimensions exhibits this type of relationship between two observations. This is an important relationship as it shows which dimensions need to be abstracted (i.e., rolled-up) in order for two observations to become comparable and/or relatable. For this reason, we define two notions of containment, namely *full containment* and *partial containment*. Full containment is exhibited when all dimension values of one observation are subsumed by the values of the respective dimensions of another observation, while partial containment is exhibited when at least one, but not all dimension values are subsumed from one observation to another.

More specifically, we define the existence of containment as a function $cont : O \times O \rightarrow [0, 1]$, where a value of 0 means that no containment relationship exists, while a value of 1 means that there exists absolute containment between a pair of observations. When $cont(o_i, o_j) = 1$, we call this *full containment*. On the other hand, when $0 < cont(o_i, o_j) < 1$, we call this *partial containment*. These are defined as follows.

Definition 6.1. ***Full Containment.***

Let o_a and o_b be two observations from datasets $D_a = (O_a, CS_a)$ and $D_b = (O_b, CS_b)$ respectively, with $CS_a = (\mathbf{P_a}, \mathbf{M_a})$ and $CS_b = (\mathbf{P_b}, \mathbf{M_b})$. Full containment between two observations, $o_a \in O_a$ and $o_b \in O_b$, exists when the following conditions hold:

$$\mathbf{P_a} \cap \mathbf{P_b} \neq \emptyset \tag{3}$$

$$\forall P_i \in \mathbf{P_a} \cap \mathbf{P_b} : h_a^i \succ h_b^i \tag{4}$$

Furthermore, the non-shared dimensions must not provide any further specialization, as stated in condition (2). When all conditions are true, the pair of observations exhibits full containment. Therefore, $(2) \wedge (3) \wedge (4) \Rightarrow cont(o_a, o_b) = 1$. We denote this with $o_a \overset{f}{\succ} o_b$. The intuition behind these conditions relies on several facts. An observation o_a fully contains o_b when values of all shared dimensions for o_a are hierarchical ancestors of the values for the same dimensions in o_b as stated in (4). Furthermore, the conjunction of the two dimension

sets must be non-empty, as stated in (3). This condition is needed because the universal condition in (4) would be evaluated to true in the case that there are no shared dimensions. Observe that the containment property is not symmetric, i.e., given $o_a \overset{f}{\succ} o_b$, then $o_b \overset{f}{\succ} o_a$ is not implied. In the example, o_{21} fully contains o_{32} and o_{34}.

Definition 6.2. ***Partial Containment.***

Let o_a and o_b be two observations from datasets $D_a = (O_a, CS_a)$ and $D_b = (O_b, CS_b)$ respectively, with $CS_a = (\mathbf{P_a}, \mathbf{M_a})$ and $CS_b = (\mathbf{P_b}, \mathbf{M_b})$. Then, o_a partly contains o_b when there exists at least one dimension whose value for o_a is a hierarchical ancestor of the value of the same dimension in o_b, as stated in the following condition:

$$\exists P_i \in \mathbf{P_a} \cap \mathbf{P_b} : h_a^i \succ h_b^i \tag{5}$$

Thus, partial containment is a generalized case of full containment. We denote this as $o_a \overset{p}{\succ} o_b$. In the example, observation o_{21} partially contains o_{31}, because *Greece* contains *Athens* but *2001* does not contain *2011*. The notation is summarized in Table 1.

Problem Definition. Based on the above, our problem is formulated as follows. Given a set D of source datasets, and a set O of observations in D, for each pair of observations $o_i, o_j \in O, i \neq j$, assess whether a) $o_i \overset{f}{\succ} o_j$, b) $o_i \overset{p}{\succ} o_j$ and c) $o_i \overset{c}{=} o_j$. In the following section, we provide our techniques for computing these properties.

Table 1: Notation

Notation	Description
o_i	The i-th observation in a set O
$\mathbf{P}$	A set of dimension schemas
$\mathbf{M}$	A set of measure schemas
P_i	The i-th dimension in a set $\mathbf{P}$
M_i	The i-th measure in a set $\mathbf{M}$
h_a^i	Value of dimension P_i for observation o_a
$h_a^i \succ h_b^i$	h_a^i is a parent of h_b^i
h_{root}	The root value *All*
$compl(o_a, o_b)$	Complementarity function
$cont(o_a, o_b)$	Containment function
$o_a \overset{c}{=} o_b$	o_a complements o_b
$o_a \overset{f}{\succ} o_b$	o_a fully contains o_b
$o_a \overset{p}{\succ} o_b$	o_a partially contains o_b

4. Algorithms for computing complementarity and containment

In this section, we present four methods for the computation of the proposed relationships, i.e, full/partial containment and complementarity. We first present a baseline method that requires quadratic computations, i.e., comparisons for all pairs of observations, and then we propose three efficient and scalable alternatives. The first uses clustering to group related observations together and limit comparisons within clusters, the second uses the notion of a cube mask lattice [27] in order to take advantage of the hierarchical relationships between the levels of the dimensions of each observation and limit comparisons between hierarchically related containers, and the third proposes an optimization over the cube masking method.

4.1. Baseline

First, we present a *baseline* method, which performs comparisons between all pairs of observations in the input dataset. Performing all pair-wise comparisons makes the *baseline* algorithm quadratic, and thus not efficient for large datasets. However, this method requires minimal preprocessing and is thus suited for smaller input sizes.

Representation. The *baseline* algorithm works under the premise that observations are represented as bit vectors in a large bitmap, which essentially defines a multidimensional feature space. Let $D_1 = (O_1, CS_1), \ldots, D_n = (O_n, CS_n)$ be n input datasets, then this bitmap is represented by an *occurrence matrix* **OM**, where each row is defined by an observation key in $\bigcup_{i=1}^{n} O_i$, and each column represents a specific value in the codelist hierarchies of the union of all dimension schemas $P = \bigcup_{i=1}^{n} P_i$, with $P_i \in CS_i$. That is, for each observation, we set the bits that correspond to the dimension values of the observation. This representation also captures the ancestral relationships between hierarchical values of the dimensions, by encoding the occurrence of a dimension value together with all of its parents. For this, we set the value of 1 to all columns that are ancestors of this value.

Prior to creating the representation space, it is often an implicit requirement of the input to perform dimension alignment, and have a reconciled dimension bus in the multidimensional space. This can be achieved by applying established entity resolution techniques for interlinking dimension values across different datasets. Even though we use the inherent linkage of Linked Open Data, it is often necessary to further resolve disambiguations and similarities in the data. Note however that Entity resolution tasks are beyond the scope of this work, which focuses on data analytics rather than on data integration problems. Thus, we consider that schema alignment and mapping of values is feasible and amortized over time (especially when data is collected from already processed sources at a regular basis) .

The occurrence matrix **OM** is a matrix that is defined over the union of all input datasets, and encodes each observation with respect to the values, all the way to the root, of its dimensions. Thus, presence of a dimension value is denoted with the corresponding bit of the column of the dimension set to 1.

Furthermore, hierarchical occurrence is also represented in **OM**, by setting all parents of the dimension value to 1, up to the root.

Each observation is defined over this matrix of dimensions $|O| \times |L|$ as a bit vector representing the occurrences of codelist values in their respective dimensions, that is, each value $h_i \in L$ becomes a feature, i.e., a column in **OM**. For example, given an observation o_a, and its value $h_a^{P_j}$ for dimension P_j, then the value as well as its hierarchical subsumption is encoded in **OM** by assigning a set bit in the column that represents $h_i = h_a^j$, as well as all of the parents of h_i. Finally, we set the columns representing h_{root} for all observations that do not contain P_j in their schema. This means that dimensions not appearing in a cube schema are assigned the *top* concept, this way marking the distinct lack of specialization in the absence of a dimension.

Conceptually, **OM** can be vertically partitioned into a series of sub-matrices, each one representing one dimension in the unified schema of the input, i.e., $\mathbf{OM} = [\mathbf{OM_1}, \ldots, \mathbf{OM}_{|\mathbf{P}|}]$, where $\mathbf{OM}_i$ is a sub-matrix that represents occurrences for all values of dimension p_i. For the example of Figure 2, and given the hierarchical code lists shown in Figure 1, the **OM** matrix is depicted n Table 2. The baseline algorithm uses **OM** for computing containment scores, and encoding these scores in a pair-wise *containment matrix*. The latter is used for the computation of both the complementarity and the containment relationships.

Table 2: Matrix OM for the example of Figure 2

	refArea										**refPeriod**					**sex**		
	WLD	EUR	AM	GR	IT	Ath	Rom	US	TX	Aus	ALL	2001	2011	Jan11	Feb11	T	F	M
obs_{11}	1	1	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	0
obs_{12}	1	0	1	0	0	0	0	1	1	1	1	0	1	0	0	1	0	1
obs_{21}	1	1	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	0
obs_{22}	1	1	0	0	1	0	0	0	0	0	1	0	1	1	0	1	0	0
obs_{31}	1	1	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	0
obs_{32}	1	1	0	1	0	1	0	0	0	0	1	0	1	1	0	1	0	0
obs_{33}	1	1	0	0	1	0	1	0	0	0	1	0	1	0	1	1	0	0

Table 3: (a) Matrix CM_1 for dimension refArea of the example of Figure 1, (b) Matrix OCM for the example of Figure 1

(a)

	obs_{11}	obs_{12}	obs_{21}	obs_{22}	obs_{31}	obs_{32}	obs_{33}
obs_{11}	1	0	0	0	1	1	0
obs_{12}	0	1	0	0	0	0	0
obs_{21}	1	0	1	0	1	1	0
obs_{22}	0	0	0	1	0	0	1
obs_{31}	1	0	0	0	1	1	0
obs_{32}	1	0	0	0	1	1	0
obs_{33}	0	0	0	0	0	0	1

(b)

	obs_{11}	obs_{12}	obs_{21}	obs_{22}	obs_{31}	obs_{32}	obs_{33}
obs_{11}	1	0	0.33	0.33	1	0.66	0.33
obs_{12}	0.33	1	0.66	0.66	0.33	0.66	0.66
obs_{21}	0.66	0.33	1	0.66	0.66	1	0.66
obs_{22}	0.33	0	0.33	1	0.33	0.66	0.66
obs_{31}	1	0	0.33	0.33	1	0.66	0.33
obs_{32}	0.66	0	0.33	0.66	0.66	1	0.33
obs_{33}	0.33	0.33	0.66	0.33	0.33	0.33	1

Specifically, for each pair of observations, or rows in **OM**, we calculate a score that denotes containment. This score is basically a normalized indicator of how many dimensions exhibit subsumption between the (ordered) pair of observations. Given a specific dimension p_i, the $|O| \times |O|$ matrix that is computed for all pairs of observations is called the *containment matrix* for dimension p_i. To compute a containment score for a pair of observations with respect to a

specific occurrence matrix $\mathbf{OM_i}$ of dimension $p_i = (L_i, \rightarrow)$, as per the definition of Section 2, we define a conditional boolean function $s_f : B^k \times B^k \rightarrow B$, where B is the boolean realm, i.e., $B = \{0, 1\}$, and B^k represents the set of bit vectors of size $k = L_i$. Assuming there is a transformation $b_v : O \rightarrow B^k$ that transforms the values of an observation to its respective bit vector for a particular dimension, then, s_f is defined as follows:

$$s_f(o_a, o_b) \mid_{\mathbf{OM_i}} = \begin{cases} 1, & \text{if } b_v(b) \subseteq b_v(a) \\ 0, & \text{otherwise} \end{cases}$$

In other words, containment exists if the bit vector of the right-hand observation is a subset of the bit vector of the left-hand observation. This can easily be computed as a logical AND operation between the bit vectors of the rows, i.e., if $b_v(b) \wedge b_v(a) = 1$, then $b_v(b) \subseteq b_v(a)$. We apply sf for o_a and o_b for dimension p_i in $\mathbf{OM_i}$. Application of this function for each dimension returns a set of $|P|$ containment matrices, $\mathbf{CM_1}, \ldots, \mathbf{CM_k}$. Adding these matrices yields the *Overall Containment Matrix* **OCM**:

$$\mathbf{OCM} = \sum_{i=1}^{k} \mathbf{CM_i}$$

The values in **OCM** are normalized between $[0, 1]$, with 0 denoting absence and 1 denoting presence of containment in all involved dimensions. This means that *full containment* between a pair of observations is derived when a cell has a value of 1, and *partial containment* when a cell has a value between 0 and 1 (non-inclusive). To assert which particular dimensions exhibit containment in a partial relationship, we examine the cells in $\mathbf{CM_i}$ being equal to 1. The occurrence of a 0 value indicates that full containment and complementarity can not hold. Note also that measure overlaps can be easily detected with a simple lookup. The construction of the **OCM** matrix is explained in Algorithm 1 *computeOCM*. We then calculate containment and complementarity using the **OCM**-based Algorithm 2 *baseline*.

Computation of complementarity. Recalling the definition of complementarity, and specifically condition (1), we can take advantage of the reflexivity of complementarity and assert that two observations are complementary when (1) and (2) hold bi-directionally. Specifically, the existence of two equal values c_i, c_j implies that there exists a bi-directional hierarchical ancestry relationship, i.e., $c_i \succ c_j$ and $c_j \succ c_i$. In this context, if two observations are related with bi-directional full containment, then they are asserted to be complementary. Therefore, during the same process of computing containment, we can also compute the complementarity relationships. For this reason, we use **OCM** to assess whether a pair of observations exhibits full containment in both directions, i.e. $o_a \overset{f}{\succ} o_b$ and at the same time $o_b \overset{f}{\succ} o_a$. For example, in Table 3, the *obs11* and *obs31* are complementary, whereas the *obs21* and *obs32* are not complementary although exhibit full containment.

Algorithm 1: *buildContainmentMatrix*

Data: An occurrence matrix **OM**, a set P of dimensions and their start indices in **OM**
Result: An *overall containment matrix* **OCM**

```
initialize OCM;
for each p_i ∈ P do
    initialize CM_pi;
    for each pair o_j, o_k ∈ OM_pi do
        if o_j AND o_k == o_j then
            CM_pi[j][k] ← 1;
        else
        end
        OCM[j][k] ← OCM[j][k] + (CM_(p_i)/|P|);
    end
end
```

Complexity Analysis. Building the containment matrix requires n^2 iterations over the full observation vectors, where n is the number of observations in the input. Even though Algorithm 1 iterates over observations $|\mathbf{P}|$ times, one for each dimension, the input for each iteration is a subset of the observation, as we take into account only the dimension values for the particular dimension. Hence, assuming that the size of a bit vector is b for all $|\mathbf{P}|$ dimensions, and $b = \sum_{i=1}^{|\mathbf{P}|} b_i$, where b_i is the size of the bit vector for dimension P_i, then the number of checks with respect to the bit vector size for dimension P_i is $b_i n^2$. Adding these for all dimensions, $\sum_{i=1}^{|\mathbf{P}|} b_i n^2 = b_1 n^2 + \cdots + b_{|\mathbf{P}|} n^2 = n^2 \sum_{i=1}^{|\mathbf{P}|} b_i = n^2 b$. Therefore, for fixed vectors of size b, the total complexity of this step is $O(n^2)$ for n observations. Then, we iterate once again all of the pairs of observations in Algorithm 2. Therefore, the total iterations required by the baseline approach are $2n^2$, with an asymptotic time complexity of $O(n^2)$.

The *baseline* algorithm operates by applying all possible pair-wise comparisons between observations in the input datasets. Thus, given n observations, the complexity is $O(n^2)$. However, during the iteration of the set of **CM** matrices, if a 0 is found at any point, we can skip further computation of full containment and complementarity, because the pair under comparison is no longer candidate for these relationships, per their definitions.

Storage-wise, **OM** needs $n \times |P|$ space for n observations and $|P|$ dimension properties in the input, following a multi-dimensional array approach. However, in our implementation, a sparse matrix implementation is adopted in order to reduce the space complexity.

4.2. Computation with Clustering

The baseline approach requires n^2 comparisons and thus quickly becomes inefficient for large datasets as it fails to scale as a result of this complexity.

Algorithm 2: *baseline*

Data: An overall containment matrix **OCM**.
Result: S_F, S_p, S_c sets of full, partial containment and complementarity relationships, and a map of partial containment relationships map_P with the dimensions they exhibit containment in.

```
1  initialize S_F, S_p, S_c;
2  for each pair o_j, o_k ∈ OCM do
3  |  if OCM[i][j] == 1 then
4  |  |  S_F ← S_F ∪ (o_i, o_j);
5  |  |  if OCM[j][i] == 1 then
6  |  |  |  S_C = S_C ∪ (o_i, o_j);
7  |  else if OCM[i][j] > 0 then
8  |  |  S_P = S_P ∪ (o_i, o_j);
9  |  |  for each p_i ∈ P do
10 |  |  |  if CM_pi[i][j] == 1 then
11 |  |  |  |  map_P(o_i, o_j, p_i) = true
12 |  |  end
13 |  else
14 |  |  continue;
15 |  end
16 end
```

The first proposed alternative method aims at improving performance by reducing the search space and executing fewer comparisons between observations. It is based on pre-clustering the input observations based on their distances in the multidimensional space, and limiting the comparisons between observations that belong to the same cluster. This approach is shown in Algorithm 3. The occurrence matrix **OM** is the input of the algorithm, and all rows are clustered into smaller occurrence matrices (Line 1). Then, the algorithm iterates through each of these clusters and applies the *buildContainmentMatrix* and *baseline* algorithms to each separate cluster (Lines 3-5). At each step of the iteration, the return arrays are updated to include the newly retrieved relationships (Line 6).

Notes on the Clustering Step. In our experiments, we employed three clustering algorithms, namely k/x-means [46], agglomerative clustering and fast canopy clustering [36]. The input for the distance function of the clustering step is a vector with the dimension values of the row. While more features such as other semantic and RDF metadata can be taken into account, previous related work [6] has shown that simple hierarchical distances of the values of the hierarchy are adequate to characterize the distance between dimension values. It is out of scope to find the optimal clustering approach for the computation of the relationships, as finding the optimal clustering parameterization or a close approximation is a non-trivial task. More sophisticated clustering approaches can be employed, however we base our selection on evaluating our approach

Algorithm 3: *baselineWithClustering*

Data: An occurrence matrix **OM**
Result: S_F, S_p, S_c sets for fully, partial containment and complementarity relationships.

```
clusters ← cluster(OM);
initialize OCM;
for i = 1 to clusters.size do
    OCM_i ← buildContainmentMatrix(clusters[i], P);
    S_Fi, S_Pi, S_Ci ← baseline(OCM_i);
    S_F, S_P, S_C ← (S_F, S_P S_C) ∪ (S_Fi, S_Pi, S_Ci);
end
return S_F, S_P, S_C;
```

on three representative clustering algorithms, a centroid-based (k/x-means, a hierarchical (agglomerative) and a fast pre-clustering approach (fast canopy). In order to optimize the pre-processing step of creating the clusters and assigning points to them, we first cluster a small sample of the data (in our experiments 10% of the input size), then we assign the rest of the input to the created clusters.

Complexity Analysis. Time and space complexity of the clustering step depends on the complexity of the chosen clustering algorithm, the number of clusters and the distribution of observations in the clusters. The baseline algorithm will run times equal to the number k of clusters. However, the distribution of observations in clusters is not known for a given collection of datasets. In the centroid-based case (canopy, k/x-means), assuming an equal distribution of $\frac{n}{k}$ observations per cluster, then the time complexity for each cluster is $\Theta(\frac{n}{k})^2$ thus making the total time complexity $\Theta(\frac{n^2}{k})$. Following a rule of thumb where $k = \sqrt{\frac{n}{2}}$, this becomes $\Theta(n^{1.5})$, at the cost of information loss, as will be shown in the experiments. This does not, however, account for the complexity of the actual clustering step, which in general is a hard problem of at least quadratic nature (e.g., hierarchical clustering requires $n^2 logn$ steps, while k-means can be solved in n^{dk} steps when the number of dimensions d and the number of centroids k are fixed).

4.3. Computation with Cube Masking

In this section, we present an alternative pre-processing method that enables flexible processing and identification of the containment and complementarity relationships in the data. The method is based on the notion of *cube masks*, which are structures that represent a fixed *level* instantiation of all the dimensions, derived from the observations in $|D|$, and the cube lattice, which represents the cube masks and their interrelationships into a graph lattice. Following, we provide the definition of these notions.

Definition 7. ***Cube Masks.***

Given a globally fixed dimension ordering, a cube mask c_i is a tuple $c_i = (l_{p_1}, l_{p_2}, \ldots, l_{p_n})$, where $p_1 \ldots p_n$ are dimensions in P, and l_{p_k} is an integer denoting the level of dimension p_k as defined in c_i. Note that P is an ordered set, and the set of all cube masks in a dataset D is denoted with C_D. In order to derive cube masks from a given dataset D, we use a function $l : C \rightarrow \mathbb{N}$ that maps codelist values to the hierarchy level they belong, and a function $mask : O \rightarrow C_D$ that maps an observation to a specific cube mask. Given the above, the mask for an observation o_i is given as:

$$mask(o_i) = (l(h_1^i), l(h_2^i), \ldots, l(h_k^i)) \tag{6}$$

A cube mask can be used as a container structure that holds references to all the observations that exhibit this level signature. In this sense, a cube mask container $||c_i||$ can be defined as the set of all observations for which the evaluation of the $mask$ function is equal to c_i, i.e.:

$$||c_i|| = \bigcup_{i=1}^{n} o_i, mask(o_i) = c_i \tag{7}$$

Each observation is assigned to exactly one cube mask. This means that for a given dataset D with k cube masks, the set of all observations O in D is given as the union of all cube mask containers $||c_i||$, i.e., $O = \bigcup_{i=1}^{k} ||c_i||$.

Definition 8. ***Cube Lattice.***

Given a dataset D and set of cube masks C_D as defined, we can build a graph lattice [27][52], where each cube mask is a node, and each edge between two nodes denotes a direct subsumption relationship between the two nodes. In this sense, an edge in the lattice represents a difference of exactly one level in exactly one dimension between the two cube mask nodes. Formally, a cube lattice is a graph $L = (V, E)$ where $V \in C_D$ and $E \in (V \times V)$. Specifically, a directed edge between two nodes exists in L, when the two nodes exhibit pair-wise subsumption in exactly one dimension, with all other dimension levels being equal, i.e. the following is true:

$$\begin{aligned} E(c_i, c_j) \in L \iff & \exists p_k \in P : l_{p_k}^i = l_{p_k}^j + 1, \\ & \forall p_m \neq p_k : l_{p_m}^i = l_{p_m}^j \end{aligned} \tag{8}$$

Furthermore, given two cube masks c_i and c_j, the relationship $c_i \succ_{cube} c_j$ is used to denote that for all dimensions in P, c_i is defined in a level that is the same or higher than c_j, and c_i is thus a hierarchical ancestor of c_j. Formally, this means that there exists a directed path in L between c_i and c_j, or that there exists a sequence of vertices $path_{ij} = (c_i = v_1, v_2, \ldots, v_k = c_j)$ such that $(v_m, v_{m+1}) \in E$ for $1 \leqslant m < k$. This further entails that each directed pair of nodes from $path_{ij}$ exhibits pair-wise subsumption, i.e. given a path $path_{ij} = (c_i = v_1, v_2, \ldots, v_k = c_j)$, then $\forall m, n \in [1 \ldots k], m < n \rightarrow v_m \succ_{cube} v_n$.

An example lattice for the three hierarchies of Figure 1 can be seen in Figure 4. Using the lattice, we can immediately prune out comparisons between observations that belong in cube masks that are not hierarchically related. For instance, in the lattice of the figure, it is unnecessary to compare $||c_{020}||$ with $||c_{310}||$ for full containment, because the relationship $c_{310} \succ_{cube} c_{020}$ is not satisfied, i.e., while the cube with signature 310 is defined on a higher level for dimensions *refArea* and *sex* ($3 \geqslant 0$ and $0 \geqslant 0$ respectively), the same is not true for dimension *refPeriod* ($1 \not\geqslant 2$). Thus, it is guaranteed that no full containment relationships can be found between $||c_{020}||$ and $||c_{310}||$.

Lattice Creation. The multidimensional cube mask lattice L can be created with one scan on the observations of D, by applying the *mask* function on each observation and storing the hash signatures of the unique cube masks into an appropriate data structure, such as a hash map. Furthermore, during the same iteration, we can derive the set of all $||c_i||$ in D. This process can be seen in Algorithm 4. First, we initiate a single scan through all observations (Line 2), then for each observation we apply the *mask* function in order to derive the cube mask of the observation (Line 4), and finally we add the observation to $cubeMaskMap$ with the found mask as key (Line 5). Then, we iterate through all the detected cubes in a nested loop and check for subsumption between the cubes (Lines 7-13).

Algorithm 4: *latticeCreation*

Result: A mapping of observations to unique cube masks, and an adjacency list with the edges between hierarchically related cubes.

```
1  initialize cubeMaskMap, lattice;
2  for each o_i ∈ O do
3  |  initialize cube;
4  |  cube ← mask(o_i);
5  |  cubeMaskMap.put(cube, cubeMaskMap.get(cube).add(o_i)) ;
6  end
7  for each c_i ∈ cubeMaskMap.keys() do
8  |  for each c_j ∈ cubeMaskMap.keys() do
9  |  |  if c_i ≻_cube c_j then
10 |  |  |  lattice.get(c_i).add(c_j)
11 |  |  end
12 |  end
13 end
14 return cubeMaskMap, lattice;
```

Baseline Computation using the Lattice. In the cases of full containment and complementarity, we do not need to compare observations that belong to lattice nodes that are not hierarchically related, such as node "121" with node "311". In the case of partial containment we look for at least one

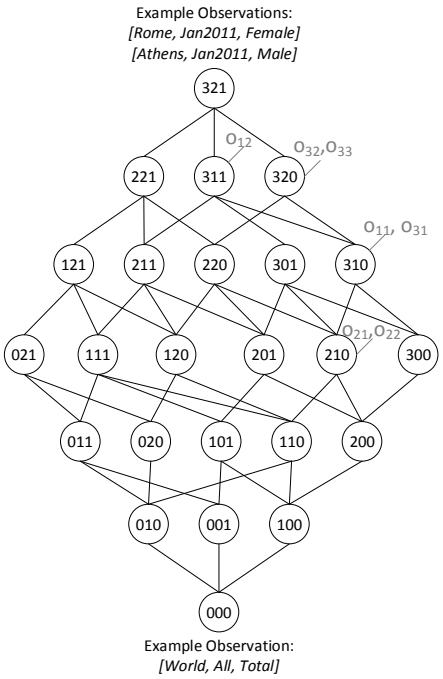


Figure 4: The lattice for the three hierarchies of Figure 2. Observations in Figure 1 are mapped to the appropriate node. The number in each node corresponds to the level of each dimension.

dimension inclusion (i.e. path) in the lattice before comparing the contents.

Based on these observations, we propose the *cubeMasking* algorithm (Algorithm 5). The algorithm first identifies cubes in the input datasets and populates the lattice, mapping observations to cubes (Line 2). Then, it iterates through cubes (Line 3) and does a pair-wise check for the cube containment criterion (Line 5). Finally it compares observations between pairs of cubes that fulfils this criterion (Lines 9-11). In order to perform these steps, we use a hash table to ensure that a value's level can be checked in constant time. We then go on to identify the cubes and build the lattice by iterating through all observations and extracting their unique combinations of dimensions and levels. To do so, we apply a hash function on each observation that both identifies and populates its cube at the same step. Finally, we iterate through the identified cubes and by doing a pair-wise check for the containment and complementarity criteria, all meaningful observation comparisons are identified. This can be seen in Algorithm 5.

alysis. In this approach, only the observations between comparable cubes are compared for the candidate relationships; the multidimensional lattice ensures that the contents of cube masks that are not hierarchically related will not be iterated quadratically. Thus, in the worst case, the maximum number of cube masks is defined as the number of permutations of dimensions and levels,

Algorithm 5: *cubeMasking*

Data: A list C with all code list terms as they appear in the datasets, a hash table *levels* with a mapping of hierarchical values to their levels, and a list O observations

Result: S_F, S_p, S_c sets for full, partial containment and complementarity

```
initialize cubeMaskMap;
cubes, lattice ← latticeCreation();
for each pair c_i, c_j ∈ cubes do
    for each p_i ∈ P do
        if not(cube_j.p_i ≺ cube_k.p_i) then
            break
        for each o_i ∈ cube_j do
            for each o_j ∈ cube_k do
                SF[o_i, o_j] ← checkFullContainment(o_i, o_j);
                SP[o_i, o_j] ← checkPartialContainment(o_i, o_j);
                SC[o_i, o_j] ← checkComplementarity(o_i, o_j);
            end
        end
    end
end
return cubeMaskMap;
function checkFullContainment
    for each p_i ∈ P do
        if not isParent(o_i.p_i, o_j.p_i) then
            return false;
        else return true;
    end
function checkPartialContainment
    for each p_i ∈ P do
        if isParent(o_i.p_i, o_j.p_i) then
            return true;
        else return false;
    end
function checkComplementarity
    if checkFullContainment(o_i.p_i, o_j.p_i) &&
     checkFullContainment(o_j.p_i, o_i.p_i) then
        return true;
    else return false;
```

i.e. $k^{(|P|)}$, where k is the maximum level of all hierarchies and $|P|$ is the number of dimensions. In order to check for comparable pairs of cube masks, we need to identify if two cube masks belong in the same ancestral path. Thus, a full (directed) traversal of the lattice is required, starting from the root. The com-

plexity is equal to the number of vertices, i.e., $k^{(|P|)}$. In the worst case, there will exist only one mask containing all observations, and all pairs of observations will have to be compared, thus still making this approach a quadratic one. However, assuming that there exist $k^{(|P|)} > 1$ vertices, with c comparable mask pairs found in the traversal, and an average of $p << n$ observations per mask, then the algorithm will require cp^2 comparisons instead of n^2, with a total cost of $O(k^{(|P|)} + cp^2)$. The reason we expect $p << n$ is that we assume a distribution of all input observations in a tractable and small number of cube masks. In real world cases, the cube schema is usually defined beforehand and populated by observations. Furthermore, the number of cube masks is restricted by the number and combinations of hierarchical levels in the input dimensions. As there are usually significantly more dimension values than hierarchy levels, we expect that all observations will be distributed to a small number of cube masks. This is also discussed in the experiments, where it can be seen in Figure 9 that as the number of input observations increases, the rate of new cube masks with respect to the input size decreases.

4.4. Optimized Cube Masking

In order to further optimize the computation of the containment and complementarity relationships, we can take advantage of the relative differences in dimensions between cube masks compared in the lattice structure that was proposed in Algorithm 5. Specifically, given two cube masks c_i and c_j, where $c_i \succ_{cube} c_j$, we can take advantage of the relative difference in dimension levels between c_i and c_j in order to define a mapping function that, given an observation $o_a \in ||c_i||$, will output the signature of the potential parent observation o_b, for which it holds that $mask(o_b) = c_j$. The requirement for this hash function is that the relative difference in dimensions and their levels between c_i and c_j is explicitly known. Formally, assume that there exists a function $L_{diff} : C_D \rightarrow C_D$ that is defined as follows:

$$L_{diff}(c_i, c_j) = (l^i_{p1} - l^j_{p1}, l^i_{p2} - l^j_{p2} \dots l^i_{pk} - l^j_{pk}) \tag{9}$$

where $p_1 \dots p_k$ are the dimensions, and it holds that $c_i \succ_{cube} c_j$. L_{diff} creates a *level difference mask* between the parent and child cubes, essentially capturing the level distance for each dimension p_i between c_i and c_j. The result of L_{diff} will be a level mask that represents the distances in levels between the dimensions of the two cube masks. For instance, consider cube masks c_{321} and c_{111} in Figure 4. Then, $L_{diff}(c_{321}, c_{111}) = (2, 1, 0)$, which means that the two cube masks have a difference of two levels in the $refArea$ dimension, a difference of one level in the $refPeriod$ dimension, and a difference of zero levels in the sex dimension. We denote the level difference mask between c_i and c_j as L^j_i.

With the use of the L_{diff} function, we can further define a function $H_{diff} : O \rightarrow O$, that, given an observation o_a and a level difference mask L^j_i, i.e., the output of an instance of L_{diff}, H_{diff} will output the potential observations that are parents of o_a and also conform to the parent cube mask of the L_{diff}

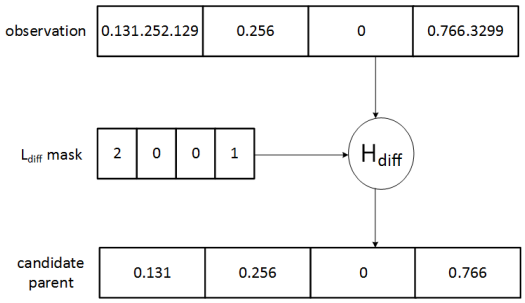


Figure 5: The lattice for the three hierarchies of Figure 2. Observations in Figure 1 are mapped to the appropriate node. The number in each node corresponds to the level of each dimension.

function, i.e., c_i. This function is defined as follows:

$$H_{diff}(o_a)|_{L_{diff}(c_i,c_j)} = \mathbf{O_a} \tag{10}$$

where $\mathbf{O_a} = \bigcup_{i=1}^{m} o_i$ such that for all i, $o_i \overset{f}{\succ} o_a$, and $mask(o_i) = c_j$, or $o_i \in ||c_j||$. In other words, H_{diff} outputs a set $\mathbf{O_a}$ that contains all potential observations that are parents of the input observation o_a with respect to the given L_{diff} mask. An example of the application of H_{diff} can be seen in Figure 5. In the top of the figure the input observation is represented as an array of dimension value signatures, pertaining to the fixed dimension ordering. The representation of the values follows the Dewey Decimal System, with parent and child values separated by dots. In the middle of the figure, the L_{diff} mask is $(2, 0, 0, 1)$. The mask and the observation instantiate the H_{diff} function which in turn outputs a candidate parent for the input observation, where each value differs in the number of levels defined by L_{diff}.

4.5. Computation of Full Containment and Complementarity

Thus, the steps that we then follow in order to derive full containment and complementarity are as follows:

1. Find next comparable pair of cube masks c_i, c_j from the lattice
2. Compute L_i^j by applying $L_{diff}(c_i, c_j)$
3. For each observation o_a in $||c_j||$, apply $H_{diff}(o_a)|_{L_i^j}$
4. Check for existence of the output of $H_{diff}(o_a)|_{L_i^j}$ in $||c_i||$

If step 4 is successful, there is a full containment relationship between c_i, c_j. These steps are described in detail in Algorithm 6, which also includes computation of complementarity. Lines 2-4 define an iteration of all nodes in the lattice. Starting from each node, we traverse the node's children by using simple recursive pre-order traversal. In Lines 6-19 the recursive function is defined.

The condition for termination is that a node does not have any other children (Lines 7-8). The algorithm first iterates through the node where the traversal initiated and each of the node's children (Line 9). For each pair of observations, the L_{diff} masking function is applied (Line 10). Then, the algorithm iterates through the contents of the parent node (Line 11) and checks for complementary observations in the child node (Line 11-13). Then, the algorithm applies the H_{diff} function on the observations of the child cube mask (Line 14) and checks if the candidate parent (i.e., the result of H_{diff} is contained in the parent cube mask (Lines 14-17). Finally, the recursion continues in the child cube mask (Line 19).

Algorithm 6: *optimizedCubeMasking*

```
   Data: A map cubes containing cube masks and their links in the lattice
   Result: S_F, S_p, S_c sets for full containment and complementarity
 1 for each  c_i ∈ cubes do
 2 |  traverse(c_i);
 3 end
 4 function traverse(c_i)
 5 |  if cubes.getChildren(c_i) == ∅ then
 6 |  |  return;
 7 |  for each c_child ∈ cubes.getChildren(c_i) do
 8 |  |  L_i^child ← L_diff(c_i, c_child);
 9 |  |  for each o_i ∈ ||c_child|| do
10 |  |  |  if o_i ∈ ||c_i|| then
11 |  |  |  |  SC[o_i, o_j] ← 1;
12 |  |  |  candidate_parent ← H_diff(o_a)|_{L_i^child};
13 |  |  |  if candidate_parent ∈ ||c_i|| then
14 |  |  |  |  o_j ← candidate_parent;
15 |  |  |  |  SF[o_i, o_j] ← 1;
16 |  |  end
17 |  |  traverse(c_child);
18 |  end
```

In order for the computation of H_{diff} to be both feasible and efficient, we need fast access to the parents of every codelist value. For this reason, we adopt a representation for observations that can capture, with small overhead, all parents of a given hierarchical value up to the root. Under this scheme, the parents of each value are encoded within the signature of the value. For example, the value *Greece* can be represented as *All.Europe.Greece*, essentially resembling the Dewey Decimal System. Then, with simple operations we can get the parent of the value that conforms to the defined level difference.

For instance, consider the case where we are comparing cube masks c_{121} and c_{100}, and we want to apply H_{diff} on an observation $o_a \in c_{121}$, with dimension values *(All.2011, All.Europe.Greece, All.Male)*, using $L_{c100}^{c121} = (0, 2, 1)$. The

value of the first dimension ($refPeriod$) stays the same, i.e., 2011, as the level difference is 0. However, the values for dimensions $refArea$ and sex will both become All, because the level differences are 2 and 1 respectively, which are both the distances of $Greece$ and $Male$ from All respectively. Eventually, the result of H_{diff} in this case is an observation $o_a^{'} = (2011, All, All)$. By definition, the $mask$ of $o_a^{'}$ is c_{100}, however, it is not guaranteed that $o_a^{'}$ exists in the dataset. If $o_a^{'} \in ||c_{100}||$, we can derive that $o_a^{'}$ fully contains o_a, and we can mark the relationship as computed.

Complexity Analysis. With this optimization, the extra space overhead required for encoding parent values into the signature of every value in the hierarchies is traded off for a significant decrease in the required comparisons at the observation level. Following the analysis of the time complexity of the simple *cubeMasking* algorithm presented in the previous section, we assume a total of $k^{(|P|)} > 1$ cube masks. The optimized masking algorithm will require $k^{(2|P|)}$ comparisons of cube masks if all cube masks are candidates for containment, and assuming that the average amount of observations in a cube mask is $p << n$, then an iteration of all the observations in one of the two compared masks is required, thus making the total cost $k^{(2|P|)}+p$. Thus, assuming that the average number of observations is asymptotically larger than the average number of cube masks, the asymptotic cost of the optimized method is $O(p)$ for the computation of full containment and complementarity, based on these assumptions. In other words, the optimized algorithm takes advantage of the knowledge of dimension level differences in a pair of comparable cube masks in order to directly retrieve the potential parents of the existing observations, thus eliminating the need for quadratic comparisons between observations of a pair of cube masks. It should be noted, however, that in the worst case, $O(n^2)$ comparisons will still be needed if only one cube masks exists, containing all n observations.

4.6. Computation of Partial Containment

In the case of partial containment, we cannot apply directly the same steps as in full containment, because partial containment does not require the existence of hierarchical subsumption between the values of all dimensions of two observations, but instead, a non-empty subset of these (condition 3). There are two main problems that we have to overcome in the computation of partial containment using the optimized cube masking approach. First, given a set of dimensions $\mathbf{P} = \{P_1, P_2, \ldots, P_n\}$ in a cube schema, only a subset $\mathbf{P}^{'} \subset \mathbf{P}$ with $\mathbf{P}^{'} \neq \emptyset$ will exhibit this subsumption between two cube masks. The set $\mathbf{P}^{'}$ cannot be determined a priori given the lattice structure, but only when the cube masks containing the observations are compared, i.e., all combinations of cube masks must be checked, which makes this a costly procedure. Second, the values of the dimensions which lie in $\mathbf{P} \setminus \mathbf{P}^{'}$ are not expected to exhibit hierarchical relationships between the compared observations. Thus, H_{diff} cannot be applied in the form of Definition 9 for detecting partial containment.On the contrary, we must first make two cube masks comparable by identifying the dimensions that are candidates for containment, and generalize to the root value the values of the remaining dimensions.

To address these issues, we propose an alternative optimization based on the optimized cube masking approach. Given two cube masks c_i, c_j, after computing $L_{diff}(c_i, c_j)$, we isolate the *positive values* in the resulting mask, as these denote the dimensions that exhibit hierarchical subsumption between the pair of cube masks, and generalize the rest of the dimension values to h_{root}. For example, given cube masks c_{321}, c_{222}, then $L_{diff}(c_{321}, c_{222}) = (1, 0, -1)$; the only positive value is the value of the first dimension. The last two dimensions must be generalized with the root value in order for the observations in the two masks to become comparable. For this, we refine the definition of H_{diff} by introducing a variant, named $H^{'}_{diff}$, defined as follows:

$$H^{'}_{diff}(o_a, \mathbf{P}^{'})|_{L_{diff}(c_i,c_j)} = \mathbf{O^{'}_a} \tag{11}$$

where $\mathbf{O^{'}_a} = \bigcup_{i=1}^{m} o_i$ such that for all i, it holds that (i) $o_i \overset{p}{\succ} o_a$ and (ii) $h_i^j = h_{root}$, where $p_j \in \mathbf{P} \setminus \mathbf{P}^{'}$. Note that, because of the generalization of the values of the dimensions in $\mathbf{P} \setminus \mathbf{P}^{'}$, the contents of $\mathbf{O^{'}_a}$ are observations that do not exist in the c_i cube mask, but basically represent candidate parents of o_a with respect to the dimensions in $P^{'}$.

For computing partial containment, we scan both $||c_i||$ and $||c_j||$ once. For each observation in $||c_i||$, we create a set of candidate parents by generalizing the dimensions in $\mathbf{P} \setminus \mathbf{P}^{'}$, and for each observation in $||c_j||$, we calculate $H^{'}_{diff}$ on $\mathbf{P}^{'}$, generalizing as well the values of $\mathbf{P} \setminus \mathbf{P}^{'}$. This procedure outputs a set of candidates in each one of the cube mask containers. Finally, we check for candidates that exist in both of these sets. In brief, the steps for computing partial containment are as follows:

1. Iterate through *all* pairs of cube masks c_i, c_j from the lattice
2. Compute L_i^j by applying $L_{diff}(c_i, c_j)$
3. Derive $\mathbf{P}^{'}$ by isolating the positive values in L_i^j
4. For each observation o_a in $||c_i||$, generalize all values of the dimensions that are not in $\mathbf{P}^{'}$ to the h_{root} value, and store the result in set $\mathbf{O^{'}_a}$
5. For each observation o_b in $||c_j||$, apply $H^{'}_{diff}(o_b, \mathbf{P}^{'})|_{L_i^j}$ and store the result in set $\mathbf{O^{'}_b}$
6. For each entry $o^{'}_b$ in $\mathbf{O^{'}_b}$, check for existence in $\mathbf{O^{'}_a}$

These steps are shown in Algorithm 7. The algorithm iterates through all possible pairs of cube masks (Lines 1-2), as opposed to the case of full containment, where only child cube masks are traversed. Then, L_{diff} is computed (Line 3) and consequently the set of containment dimensions $P^{'}$ is derived (Lines 4-7). The algorithm iterates through the container of the outer cube mask (Line 9), and for each observation, it generalizes the dimension values to the h_{root} value (Line 10), mapping the newly created observation to the original one (Line 11).

Next, the algorithm iterates through the contents of the inner cube mask (Line 13) and applies $H^{'}_{diff}$ on the contained observations (Line 14). Each created observation is then checked for existence in the candidates of the outer set, marking the relationship as partial containment in case of success (Line 15-17).

Algorithm 7: *optimizedCubeMaskingPartial*

```
   Data: A map cubes containing cube masks and their links in the lattice
   Result: S_p set for partial containment
 1 for each c_i ∈ cubes do
 2     for each c_j ∈ cubes do
 3         L_i^j ← L_diff(c_i, c_j);
 4         for each l_p ∈ L_i^j do
 5             if l_p > 0 then
 6                 P' ← p;
 7         end
 8         initialize candidate_set_i;
 9         for each o_a ∈ ||c_i|| do
10             o'_a ← generalize(o_a, P \ P');
11             candidate_set_i.put(o'_a, o_a);
12         end
13         for each o_b ∈ ||c_j|| do
14             o'_b ← H'_diff(o_b, P');
15             if candidate_set_i.contains(o'_b) then
16                 o_a ← candidate_set_i.get(o'_b);
17                 SP[o_a, o_b] ← 1;
18         end
19     end
20 end
```

Complexity Analysis. As in the case for full containment, the extra space overhead required for encoding the candidate sets for a given pair of cube masks is traded off for a decrease in the required comparisons at the observation level. Assuming a total of $k^{(|P|)}$ cube masks, then we need to compare all pairs of cube masks for partial containment, or $k^{(2|P|)}$. For each compared pair of cube masks, we scan both cube masks and apply the $H^{'}_{diff}$ function in their contents, which makes the total cost $k^{(2|P|)} + 2p$. Asymptotically, this amounts to a linear time complexity of $O(p)$, which is still linear with respect to the average observation cardinality in the cube masks. As in the other cube masking approaches, however, in the worst case, there will only exist one cube mask containing the whole set of observations, which makes the complexity $O(n^2)$ for n observations. In real cases, the performance improvements are still significant and fall under the assumptions made in these analyses.

5. Experimental Evaluation

The goal of the experimental evaluation is to assess the performance of all the above methods in terms of time efficiency, accuracy and scalability of the proposed algorithms and evaluate our methods in comparison with inference and SPARQL query processing techniques widely used for detecting relationships in RDF data. We demonstrate that our approach achieves small execution time in commodity hardware and outperforms traditional techniques, which fail to scale up as the number of observations increase. In the following sections, we first provide the experimental setting, i.e., the datasets used for the experiments, the metrics and the setup of the experimental environment. Then we proceed with the evaluation of the proposed metrics, and we present and discuss the results.

5.1. Setting

We have selected seven real-world datasets on government and demographic statistics. The datasets were taken from Eurostat[4], the Eurostat Linked Data Wrapper[5], World Bank[6] and the linked-statistics.gr project [7]. Eurostat offers a wide variety of statistical data on countries of the EU region, and while it did not provide data in RDF format at the time of writing, some of the datasets contained therein are provided through the Eurostat Linked Data Wrapper in the RDF Data Cube (QB) representation format. World Bank is a rich source of statistical data for all countries of the world, and linked-statistics.gr is a project that offers data from the official Greek statistics authority, converted to RDF with the QB representation. In the case of datasets in the CSV format, we converted them to QB by adopting the approach of [51]. Notably, several other tools can also be used for the conversion, such as CSV2RDF[8], OpenCube[9] and Open Refine[10]. In this context, the header labels of the CSV files are converted to dimensions (each assigned with a unique URI), and each row becomes an observation. The cell values are matched automatically to existing terms in the shared code lists.

The datasets consist of a total of 11 dimensions, 6 measures and more than 2,500 unique hierarchical terms, for a total of 260,000 observation records. The seven datasets cover demographic statistical data on population, unemployment, births/deaths, national economy (GDP) and internet adoption by household number, while the dimensions cover features such as geographical region, reference periods, country of citizenship, human genre (sex), level of education, and household size. The dataset details can be seen in Table 4.

[4] http://epp.eurostat.ec.europa.eu/portal/page/portal/statistics/search_database

[5] http://estatwrap.ontologycentral.com/

[6] http://data.worldbank.org/

[7] http://linked-statistics.gr/

[8] http://www.w3.org/TR/csv2rdf/

[9] http://opencube-toolkit.eu/

[10] http://refine.deri.ie/

Table 4: Dataset dimensions, amount of observations and respective measures

Dataset (# of obs)	refArea	refPeriod	sex	unit	age	economic activities	citizenship	education	household size	measure
D_1 (58k)	Y	Y	Y	Y	Y	N	Y	N	N	Population
D_2 (4.2k)	Y	Y	N	Y	N	N	N	N	Y	Members
D_3 (6.7k)	Y	Y	Y	Y	Y	N	N	Y	N	Population
D_4 (15k)	Y	Y	N	Y	N	N	N	N	N	Births
D_5 (68k)	Y	Y	Y	Y	Y	N	Y	N	N	Deaths
D_6 (73k)	Y	Y	N	Y	N	N	N	N	N	GDP
D_7 (21.6k)	Y	Y	N	N	N	Y	N	N	N	Compensation

We preprocessed the code lists in order to align dimension and hierarchy values across the input data space, by employing LIMES [40], a Linked Data interlinking tool that is commonly used for term alignment in the LOD cloud. LIMES can be configured to use restriction rules on the input (for example, candidate matches must exhibit the same *rdf:type* values), and has a customizable distance metric parameter, that can be programmed to use combined distance functions such as aggregates (maximum, average etc.) on multiple distance functions, such as cosine similarity, jaccard distance and levenshtein distance. For our experiments, we configured LIMES to match the code list terms by comparing the string URIs, and used their cosine distance in order to discover matches based on the URI suffixes.

Metrics. The main aim of the experimental evaluation was to compare the performance of all the proposed methods with respect to *execution time*. Specifically, we measure *execution time* by including the pre-processing and computation steps for the defined relationships. Furthermore, we report the total number of observations that are accessed/compared for each method. Especially for the case of the clustering approach, we are also interested in the achieved *recall* of the computed relationships, as this method is the only one that does not guarantee 100% recall. In this sense, *recall* is defined as the ratio of correctly found relationships to the number of all relationships in the datasets, as discovered by the other methods. Note that we do not consider any decrease in recall induced by the tool used in the dimension alignment step. The alignment process is performed as a pre-processing step for all algorithms, providing the same input in each case, and is thus independent from the achieved recall.

Experimental Setup. We implement our approach in Java 1.8 on a single machine with allocated memory of 16GB DDR3. For the experiments, we gradually increased the input size, starting from 2,000 observations and increasing it with a fixed step of 20,000 observations. For the *clustering* method, we have experimented with three clustering algorithms, namely fast canopy, x-means, and hierarchical clustering, using Jaccard similarity as a distance metric, and applying them on random 10% samples of the full input. In the series of experiments, we performed comparisons between the proposed algorithms, as well as a SPARQL-based and an inference-based alternative.

SPARQL-based approach. We consider a SPARQL-based alternative approach, for which we devised three queries for the detection of the underlying

relationships of containment and complementarity. For the containment relationships, the subsumption between dimension values can occur in any level, and thus must be modelled using wildcard-enabled property paths, which are directly supported by the SPARQL 1.1 recommendation and most of its implementations. However, a requirement for the case of full containment is the universal restriction over the subsumption of the dimension values. As SPARQL does not explicitly allow universal quantification in its syntax, it has to be mimicked with the use of negation and nested recursion, which makes the query complicated to write and costly to execute. Partial containment can be easily detected, but it is complicated to derive the exact dimensions that do not exhibit containment. Given the above, and for the sake of simplicity, we design the three queries with the scope of detecting the *existence* of the relationships, and we do not quantify it like in the computation of the OCM matrix. The queries can be seen in Appendix A.

Rule-based. The rule-based approach consists of three forward-chaining rules implemented in Jena, as the Jena generic rule reasoner is simple to use and offers the required expressiveness. The rules can be seen in Appendix B.

The datasets and code are available online at `http://github.com/mmeimaris`.

5.2. Experimental Results

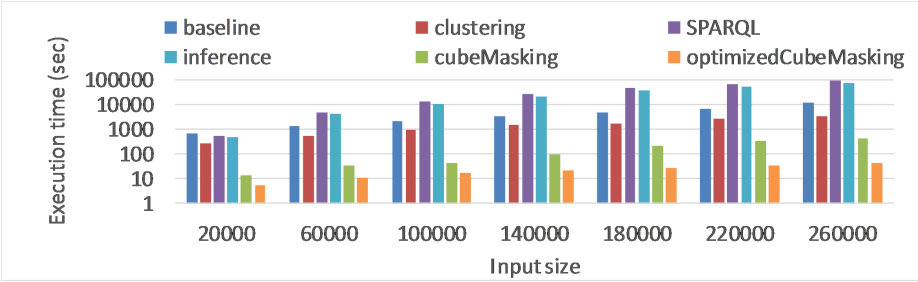
The set of conducted experiments suggests that the baseline algorithm can be improved significantly by all three of the optimized methods. In what follows, we describe the achieved results for each of the algorithms.

5.2.1. Baseline

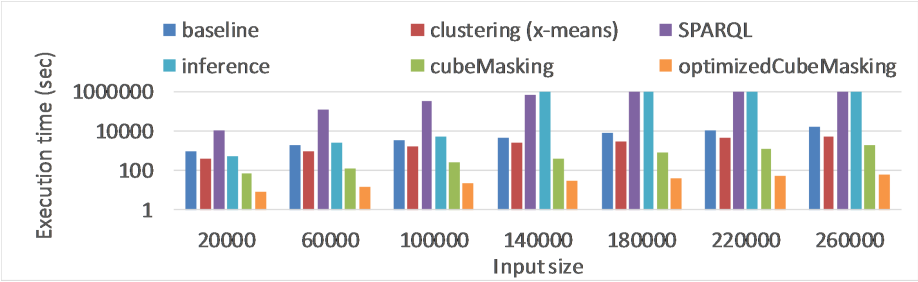
The *baseline* algorithm behaves quadratically with respect to input size, as it performs n^2 comparisons for n input observation rows. The results are shown in Figure 6(a-c). When computing full containment and complementarity, the required checks are decreased, because we can quickly skip pairs of rows that fail the subsumption criterion at least once. Furthermore, recall from Algorithm 2, that complementarity and full containment relationships are computed at the same pass, i.e., a bilateral full containment relationship between two observations implies their complementarity. The total number of observation pairs compared in the baseline method can be seen in Figure 7, and includes all possible pairs of observations in the datasets.

5.2.2. Clustering

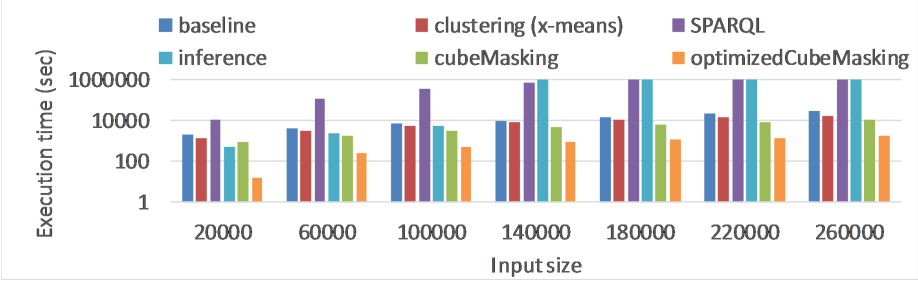
The *clustering* approach has been implemented on top of the baseline, by configuring the code to cluster the input observation rows and then perform the baseline on each cluster. In this set of experiments, we run three different settings, changing the clustering algorithm in each one. The three algorithms we have used, two centroid-based and one agglomerative, are (i) x-means, which is a variant of k-means that automatically configures the number of centroids based on the input, (ii) fast canopy clustering, and (iii) hierarchical clustering. For all approaches, we configured the system to use a sample 10% of the input



(a) Execution time (seconds) for complementarity



(b) Execution time (seconds) for full containment



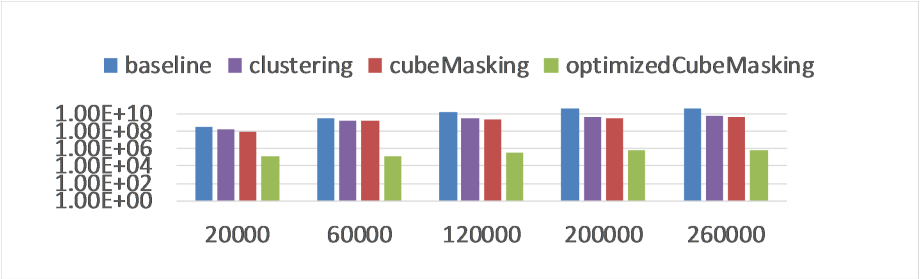
(c) Execution time (seconds) for partial containment

Figure 6: Execution performance experiments

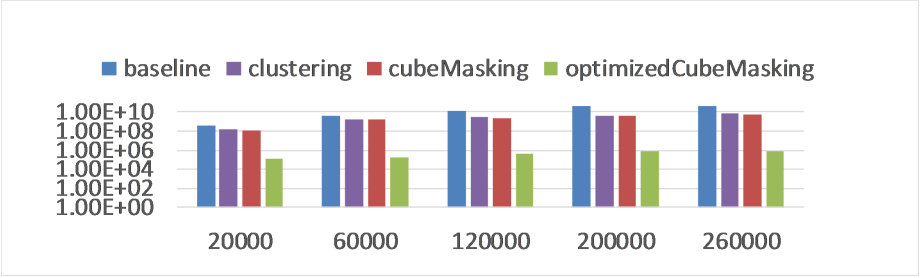
size, and then assigned the remainder of the input to the detected clusters. We achieved varying degrees of recall, which can be seen in Figure 8. According to our results, the k-means variant achieved the higher degree of recall. In Figure 6(a-c) we report the execution times with x-means, compared with the other approaches. Furthermore, the total number of compared observation pairs can be seen in Figure 7 for the cases of full and partial containment. As a general note, the clustering approach outperformed the naive baseline algorithm, despite the eventual trade-off between runtime performance and relationship recall.

5.2.3. Cube Masking

The performance of the *cubeMasking* algorithm yields a substantial improvement with respect to both the baseline and the baseline with clustering. This is



(a) Number of observation accesses for full containment



(b) Number of observation accesses for partial containment

Figure 7: Quantified accesses to individual observations for containment relationships

attributed to several factors. First, the cost of identifying the cube masks and building the lattice is linear with respect to the input size, as it requires one scan over the data. Second, the number of comparisons is significantly decreased, as comparisons are limited only between hierarchically related cube masks, while maintaining full recall. These results can be seen in Figures 6 and 7. A potential drawback of this approach is the loss of the performance advantage (i.e., the decrease of comparisons) when the number of cube masks is very large with respect to the input rows, or the distribution of the data is uneven (e.g., a small number of cube masks that cover a large percentage of the input). However, the rate of cube masks per input rows tends to converge logarithmically as the input size increases. This can be seen in Figure 9.

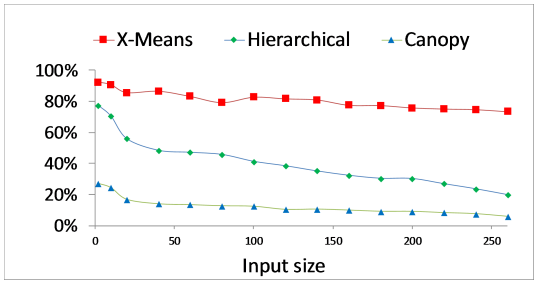


Figure 8: Achieved recall for the clustering approaches

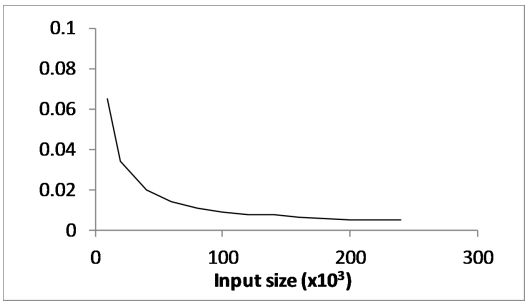


Figure 9: Rate of cube masks per row

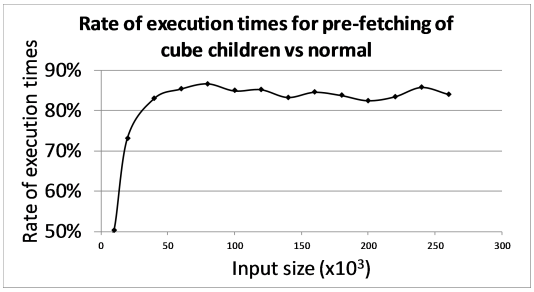


Figure 10: Execution rate (pre-fetching vs non-pre-fetching)

While *cubeMasking* operates in a similar way to the clustering method, it is more efficient because of the hierarchical relationships between the cube masks, which allow for less and more relevant comparisons between the observations. We implement the lattice as a graph data structure, so that we can have fast access to the children of each cube mask. However, we also experimented with pre-processing the lattice in order to derive the parent-child relationships and store them in memory for constant time access for each cube mask. This pre-fetching yielded a 15% improvement of the execution time, as can be seen in Figure 10, at the cost of an extra scan of the cube masks, and a minor imposed overhead in the storage of the system, for explicitly storing all paths between cube masks in the lattice (in our experiments, this was less than 1MB for the full dataset).

5.2.4. Optimized Cube Masking

The optimized cube masking algorithm yields a substantial improvement in the runtime performance for the computation of full containment, partial containment and complementarity, as can be seen in the execution times of Figure 6, as well as the total number of accessed observations in Figure 7. This is attributed to its decreased computational complexity with respect to the original cube masking approach. However, for large numbers of dimensions, the H_{diff} function becomes costlier as the observation signatures require larger numbers of hierarchy abstractions. For the case of partial containment, the

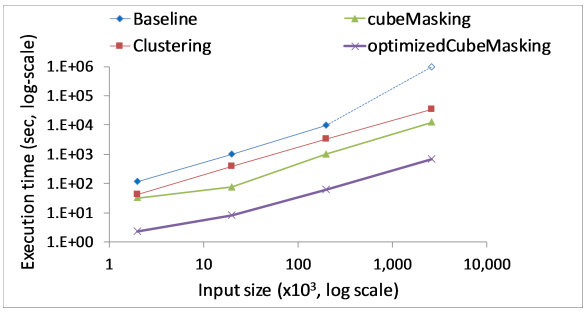


Figure 11: Execution time (log-log) with synthetic dataset

optimization exhibits a smaller relative advantage when compared to the original cube masking approach as the input size increases, as can be seen in Figure 6. However, it is still the fastest of all the approaches we experimented with.

The optimized cube masking approach is the fastest of all the tested approaches, as it takes advantage of exact hash signatures which can be checked in constant time. This is reflected in the experiments shown in Figure 6.

5.2.5. SPARQL and Rule-based

The runtime performance for these two alternatives is satisfactory for very small input sizes, as shown in Figure 6(a), (b) and (c) (less than 40k observations). However, they become intractable fast, as they either hit the time-out limits or they have vast memory requirements. This renders them non-scalable and thus not deployable in realistic settings over large real world datasets. The quality of performance is dependent on the transitive nature of the relationships, which quickly makes the search space large. The SPARQL queries timed out quickly as the number of rows increases when executed in Virtuoso (see Figure 6(c)). In the experiments the SPARQL method was still inadequate after the query relaxation described in Appendix A. For the rule based approach, the space overhead became large quickly, triggering several out of memory errors.

5.3. Scalability

We conducted a set of experiments to test the scalability of the proposed approaches, by creating a complementary synthetic dataset (x10 of the full size of the real world datasets), by following a similar approach as in [16] and extending the existing data by creating observation rows that follow a projected distribution of the data with respect to the real-world datasets. More specifically, we used a number of cube masks derived from Figure 9 for 2.5 million observations, and populated the newly created cube masks accordingly.

As expected, the experiments show that the SPARQL-based and rule-based approaches do not scale. The results for the rest of the methods are shown in Figure 11, with the dotted line representing the projection of the baseline approach, as it took more than 7 days to complete.

These results show that the *clustering*, *cubeMasking* and *optimizedCubeMasking* methods are scalable for larger input sizes, with the latter two having

a more clear advantage because of the reduced number of needed comparisons. It should be noted, however, that in some edge cases where the input contains very large numbers of cube masks with sparse and even distributions of observations in these cube masks, the *cubeMasking* and *optimizedCubeMasking* approaches will lose their relative advantage to the quadratic baseline. Such cases justify the need for probabilistic approaches such as *clustering*, especially when runtime performance is more important than the achieved recall.

6. Concluding Remarks

In this paper, we have presented and compared four approaches for discovering three types of instance-level relationships between observations of multidimensional RDF data cubes. To this end, we have formally defined *full containment* , *partial containment*, and *complementarity* between multidimensional observations. We performed an extensive experimental evaluation between the proposed approaches and with two traditional approaches, namely a SPARQL-based and a rule-based method, and we found that our algorithms outperform the traditional approaches in both execution time and scalability. As future work, we intend to tackle incremental updates of the relationships in dynamically growing datasets. Furthermore, we intend to study the possibilities of utilizing further semantic metadata in order to assist and also enrich the detection process. Finally, we plan to study the performance of the proposed methods in distributed and parallel contexts.

Acknowledgement. We thank the reviewers and the editor for their valuable comments that have improved the readability and clarity of the paper. We acknowledge support of this work by the projects "EU H2020 SlideWiki (#688095)" and "Moving from Big Data Management to Data Science"(MIS 5002437/3) which is implemented under the Action "Reinforcement of the Research and Innovation Infrastructure", funded by the Operational Programme "Competitiveness, Entrepreneurship and Innovation" (NSRF 2014-2020) and co-financed by Greece and the European Union (European Regional Development Fund).

[1] A. Abelló, J. Darmont, L. Etcheverry, M. Golfarelli, J. N. Mazón López, F. Naumann, T. B. Pedersen, S. Rizzi, J. C. Trujillo Mondéjar, P. Vassiliadis, et al. Fusion cubes: towards self-service business intelligence. 2013.

[2] A. Abelló, O. Romero, T. B. Pedersen, R. Berlanga, V. Nebot, M. J. Aramburu, and A. Simitsis. Using semantic web technologies for exploratory olap: a survey. *IEEE transactions on knowledge and data engineering*, 27(2):571–588, 2015.

[3] C. C. Aggarwal and P. S. Yu. Outlier detection for high dimensional data. In *ACM Sigmod Record*, volume 30(2), pages 37–46. ACM, 2001.

[4] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia. Similarity measures for olap sessions. *Knowledge and information systems*, 39(2):463–489, 2014.

[5] N. Alrayes and W.-S. Luk. Automatic transformation of multi-dimensional web tables into data cubes. *Data Warehousing and Knowledge Discovery*, pages 81–92, 2012.

[6] E. Baikousi, G. Rogkakos, and P. Vassiliadis. Similarity measures for multidimensional data. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 171–182. IEEE, 2011.

[7] S. Bayerl and M. Granitzer. Discovering, ranking and merging rdf data cubes. In *Semantic Computing (ICSC), 2017 IEEE 11th International Conference on*, pages 133–140. IEEE, 2017.

[8] O. Benjelloun, H. Garcia-Molina, D. Menestrina, Q. Su, S. E. Whang, and J. Widom. Swoosh: a generic approach to entity resolution. *The VLDB Journal—The International Journal on Very Large Data Bases*, 18(1):255–276, 2009.

[9] C. Böhm, F. Naumann, M. Freitag, S. George, N. Höfler, M. Köppelmann, C. Lehmann, A. Mascher, and T. Schmidt. Linking open government data: what journalists wish they had known. In *Proceedings of the 6th International Conference on Semantic Systems*, page 34. ACM, 2010.

[10] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 74–83. ACM, 2004.

[11] M. Ceci, A. Cuzzocrea, and D. Malerba. Effectively and efficiently supporting roll-up and drill-down olap operations over continuous dimensions via hierarchical clustering. *Journal of Intelligent Information Systems*, 44(3):309–333, 2015.

[12] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 503–514. ACM, 2006.

[13] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *ACM Sigmod record*, 26(1):65–74, 1997.

[14] R. Cyganiak, D. Reynolds, and J. Tennison. The rdf data cube vocabulary. *W3C Recommendation (January 2014)*, 2013.

[15] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 817–828. ACM, 2012.

[16] B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: optimizing noise sources and consistency. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 217–228. ACM, 2011.

[17] F. M. Donini. Complexity of reasoning. In *The description logic handbook*, pages 96–136. Cambridge University Press, 2003.

[18] G. Drzadzewski and F. W. Tompa. Partial materialization for online analytical processing over multi-tagged document collections. *Knowledge and Information Systems*, 47(3):697–732, 2016.

[19] V. Efthymiou, G. Papadakis, G. Papastefanatos, K. Stefanidis, and T. Palpanas. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems*, 65:137–157, 2017.

[20] L. Etcheverry and A. A. Vaisman. Qb4olap: a new vocabulary for olap cubes on the semantic web. In *Proceedings of the Third International Conference on Consuming Linked Data-Volume 905*, pages 27–38. CEUR-WS. org, 2012.

[21] L. Etcheverry and A. A. Vaisman. Querying semantic web data cubes. In *AMW*, 2016.

[22] A. Giacometti, P. Marcel, E. Negre, and A. Soulet. Query recommendations for olap discovery driven analysis. In *Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP*, pages 81–88. ACM, 2009.

[23] L. I. Gómez, S. A. Gómez, and A. A. Vaisman. A generic data model and query language for spatiotemporal olap cube analysis. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 300–311. ACM, 2012.

[24] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[25] J. Han and Y. Fu. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *KDD Workshop*, pages 157–168, 1994.

[26] J. Han, Y. Fu, W. Wang, J. Chiang, W. Gong, K. Koperski, D. Li, Y. Lu, A. Rajan, N. Stefanovic, et al. Dbminer: A system for mining knowledge in large relational databases. In *KDD*, volume 96, pages 250–255, 1996.

[27] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *ACM SIGMOD Record*, volume 25, pages 205–216. ACM, 1996.

[28] I. Horrocks, P. F. Patel-Schneider, S. Bechhofer, and D. Tsarkov. Owl rules: A proposal and prototype implementation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(1):23–40, 2005.

[29] K. C. Hsu and M.-Z. Li. Techniques for finding similarity knowledge in olap reports. *Expert Systems with Applications*, 38(4):3743–3756, 2011.

[30] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi. Optimizing aggregate sparql queries using materialized rdf views. In *International Semantic Web Conference (1)*, pages 341–359, 2016.

[31] B. Kämpgen and A. Harth. No size fits all–running the star schema benchmark with sparql and rdf aggregate views. In *The Semantic Web: Semantics and Big Data*, pages 290–304. Springer, 2013.

[32] B. Kämpgen, S. Stadtmüller, and A. Harth. Querying the global cube: Integration of multidimensional datasets from the web. In *Knowledge Engineering and Knowledge Management*, pages 250–265. Springer, 2014.

[33] T. Komamizu, T. Komamizu, T. Amagasa, T. Amagasa, H. Kitagawa, and H. Kitagawa. H-spool: A sparql-based etl framework for olap over linked data with dimension hierarchy extraction. *International Journal of Web Information Systems*, 12(3):359–378, 2016.

[34] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 275–286. VLDB Endowment, 2002.

[35] V. Markl, F. Ramsak, and R. Bayer. Improving olap performance by multidimensional hierarchical clustering. In *Database Engineering and Applications, 1999. IDEAS'99. International Symposium Proceedings*, pages 165–177. IEEE, 1999.

[36] A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. ACM, 2000.

[37] M. Meimaris and G. Papastefanatos. Containment and complementarity relationships in multidimensional linked open data. In *Second International Workshop for Semantic Statistics SemStats*, 2014.

[38] M. Meimaris, G. Papastefanatos, P. Vassiliadis, and I. Anagnostopoulos. Efficient computation of containment and complementarity in rdf data cubes. In *EDBT*, pages 281–292, 2016.

[39] P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th International Conference on Semantic Systems*, pages 1–8. ACM, 2011.

[40] A.-C. N. Ngomo and S. Auer. Limes-a time-efficient approach for large-scale link discovery on the web of data. *integration*, 15:3, 2011.

[41] S. M. Nutley, H. T. Davies, and P. C. Smith. *What works?: Evidence-based policy and practice in public services*. MIT Press, 2000.

[42] G. Papadakis, G. Alexiou, G. Papastefanatos, and G. Koutrika. Schema-agnostic vs schema-based configurations for blocking methods on homogeneous data. *Proceedings of the VLDB Endowment*, 9(4):312–323, 2015.

[43] G. Papadakis, E. Ioannou, C. Niederée, and P. Fankhauser. Efficient entity resolution for large heterogeneous information spaces. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 535–544. ACM, 2011.

[44] G. Papadakis, E. Ioannou, T. Palpanas, C. Niederee, and W. Nejdl. A blocking framework for entity resolution in highly heterogeneous information spaces. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2665–2682, 2013.

[45] G. Papadakis, G. Papastefanatos, T. Palpanas, and M. Koubarakis. Scaling entity resolution to large, heterogeneous data with enhanced meta-blocking. In *EDBT*, pages 221–232, 2016.

[46] D. Pelleg, A. W. Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *ICML*, pages 727–734, 2000.

[47] G. Piatetsky-Shapiro, R. J. Brachman, T. Khabaza, W. Kloesgen, and E. Simoudis. An overview of issues in developing industrial data mining and knowledge discovery applications. In *KDD*, volume 96, pages 89–95, 1996.

[48] M. F. Rahmany, A. Asudehy, N. Koudas, and G. Das. Efficient computation of subspace skyline over categorical domains. *arXiv preprint arXiv:1703.00080*, 2017.

[49] S. Rizzi, M. Golfarelli, and S. Graziani. An olam operator for multidimensional shrink. *International Journal of Data Warehousing and Mining (IJDWM)*, 11(3):68–97, 2015.

[50] O. Romero and A. Abelló. Automating multidimensional design from ontologies. In *Proceedings of the ACM tenth international workshop on Data warehousing and OLAP*, pages 1–8. ACM, 2007.

[51] G. Sathe and S. Sarawagi. Intelligent rollups in multidimensional olap data. In *VLDB*, volume 1, pages 531–540, 2001.

[52] A. Shukla, P. Deshpande, J. F. Naughton, et al. Materialized view selection for multidimensional datasets. In *VLDB*, volume 98, pages 488–499, 1998.

[53] E. Tambouris. Multidimensional open government data. *JeDEM-eJournal of eDemocracy and Open Government*, 8(3):1–11, 2016.

[54] K.-L. Tan, P.-K. Eng, B. C. Ooi, et al. Efficient progressive skyline computation. In *VLDB*, volume 1, pages 301–310, 2001.

[55] A. Vaisman and E. Zimányi. *Data Warehouse Systems: Design and Implementation*. Springer, 2014.

[56] J. Varga, L. Etcheverry, A. A. Vaisman, O. Romero, T. B. Pedersen, and C. Thomsen. Qb2olap: Enabling olap on statistical linked open data. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 1346–1349. IEEE, 2016.

[57] J. Varga, A. A. Vaisman, O. Romero, L. Etcheverry, T. B. Pedersen, and C. Thomsen. Dimensional enrichment of statistical linked open data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 40:22–51, 2016.

[58] B. Villazón-Terrazas, L. M. Vilches-Blázquez, O. Corcho, and A. Gómez-Pérez. Methodological guidelines for publishing government linked data. In *Linking government data*, pages 27–49. Springer, 2011.

[59] J. Volz, C. Bizer, M. Gaedke, and G. Kobilarov. Silk-a link discovery framework for the web of data. *LDOW*, 538, 2009.

[60] X. Xie, X. Hao, T. B. Pedersen, P. Jin, and J. Chen. Olap over probabilistic data cubes i: Aggregating, materializing, and querying. In *Data Engineering (ICDE), 2016 IEEE 32nd International Conference on*, pages 799–810. IEEE, 2016.

[61] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *Proceedings of the 31st international conference on Very large data bases*, pages 241–252. VLDB Endowment, 2005.

Appendix A. SPARQL Queries

Notes on the SPARQL-based approach. As it has been argued in this paper, property paths are directly supported by SPARQL 1.1 and are necessary for computing whether two values are related hierarchically. A different alternative is to compute the transitive closure of the data and materialize these relationships, however we do not address efficient materialization of transitivity in RDF datasets. Universal quantification must be mimicked by using a negation construct that includes a nested recursion. This negation actually ensures that there is no dimension between two candidate observations that does not exhibit hierarchically related values. This is useful for computing full containment. On the other hand, partial containment can be detected by SPARQL ASK queries merely by checking whether at least one occurrence of hierarchically related values is present in any of the shared dimensions between two observations.

In the case of *partial containment*, the queries for materializing and detecting pairs of observations are as follows:

Partial Containment (materialization):

```
CONSTRUCT {
        [
  rdf:type imis:PartialContainment ;
          imis:containedObservation ?o1 ;
          imis:containingObservation ?o2 ;
#       other metadata can be added about the containment here
        ]
}
WHERE {
?o1 a qb:Observation .
?o2 a qb:Observation .
?o1 ?d1 ?v1.
?o2 ?d1 ?v2.
?v1 skos:broaderTransitive/skos:broaderTransitive* ?v2

}
```

Partial Containment (detection):

```
SELECT DISTINCT ?o1, ?o2
WHERE {
        ?o1 a qb:Observation .
        ?o2 a qb:Observation .
        ?o1 ?d1 ?v1.
        ?o2 ?d1 ?v2.
        ?v1 skos:broaderTransitive/skos:broaderTransitive* ?v2
        FILTER(?o1 != ?o2)
}
```

The above query will select pairs of *?o1* and *?o2* that have at least one dimension with ancestral values; *?v1* must be a parent of *?v2*. The above query does not provide the number of dimensions that participate in the *partial containment*; this would make the query more complicated.

In the case of *full containment*, the queries for materializing and detecting pairs of observations are as follows:

Full Containment (Materialization):

```
CONSTRUCT {
        [
  rdf:type imis:FullContainment ;
          imis:containedObservation ?o1 ;
          imis:containingObservation ?o2 ;
#       other metadata can be added about the containment here
        ]
}
WHERE {
?o1 a qb:Observation .
?o2 a qb:Observation .
?o1 ?d1 ?v1.
?o2 ?d1 ?v2.
?v1 skos:broaderTransitive/skos:broaderTransitive* ?v2
        ?o1 ?d2 ?v12 .
        ?o2 ?d2 ?v22 .
        FILTER NOT EXISTS {
OPTIONAL {
                ?v12 skos:broaderTransitive/skos:
                    broaderTransitive* ?v22
}
FILTER (!BOUND(?v22))
}
}
```

Full Containment (Detection):

```
SELECT DISTINCT ?o1, ?o2 WHERE {
?o1 a qb:Observation .
?o2 a qb:Observation .
?o1 ?d1 ?v1.
?o2 ?d1 ?v2.
?v1 skos:broaderTransitive/skos:broaderTransitive* ?v2
        ?o1 ?d2 ?v12 .
        ?o2 ?d2 ?v22 .
        FILTER NOT EXISTS {
OPTIONAL {
                ?v12 skos:broaderTransitive/skos:
                    broaderTransitive* ?v22
}
FILTER (!BOUND(?v22))
}
}
```

The above queries return pairs observations, *?o1* and *?o2*, that have all dimension values exhibiting ancestral relationships; all *?v1* must be a parent of *?v2* using property paths of undefined length.

Complementarity (Materialization):

In the case of *complementarity*, we tested the data against the following SPARQL queries:

```
CONSTRUCT {
        [
  rdf:type imis:Complement;
          imis:observation ?o1 ;
          imis:observation ?o2 ;
#       other metadata can be added about the containment here
        ]
}
WHERE {
?o1 a qb:Observation .
?o2 a qb:Observation .
?o1 ?d1 ?v1.
?o2 ?d1 ?v1.
        FILTER NOT EXISTS {
                ?o1 ?d2 ?v12 .
                ?o2 ?d2 ?v22 .
                FILTER(?v12!=?v22)
        }
}
```

Complementarity (Detection):

```
SELECT DISTINCT ?o1, ?o2
WHERE {
?o1 a qb:Observation .
?o2 a qb:Observation .
?o1 ?d1 ?v1.
?o2 ?d1 ?v1.
        FILTER NOT EXISTS {
                ?o1 ?d2 ?v12 .
                ?o2 ?d2 ?v22 .
                FILTER(?v12!=?v22)
        }
}
```

The queries will return pairs of observations with no different values in their shared dimensions.

Note that in all of the queries, we have relaxed the conditions presented in section 2, as the runtimes would be even slower, and their syntax complicated.

Appendix B. Rules

Notes on the rule-based approach. The rule based approach is performed with forward-chaining rules for the cases of containment and complementarity. For *full containment*, we check for pairs of observations that exhibit both existential and universal quantification in the subsumption of their respective dimension values. The existential quantification is needed to ensure that

there exists at least one relationship, while the universal is needed to ensure that all relationships exist. The rule for *full containment* is as follows:

```
observation(o1) ∧ observation(o2)
∧ (o1 ≠ o2)
∧ ∃p.(has_dimension_value(o1,p,v1)
        ∧ has_dimension_value(o2,p,v2)
        ∧ is_ancestor(v1,v2))
∧ ∀p.( has_dimension_value(o1,p,v1)
        ∧ has_dimension_value(o2,p,v2)
        ∧ is_ancestor(v1,v2))
⇒ full_containment(o1,o2)
```

Similarly, the rule for *partial containment* checks the existential restriction; that is, we need at least one pair of dimension values to exhibit a containment relationship between o_1 and o_2. Therefore, the rule is as follows:

```
observation(o1) ∧ observation(o2)
∧ (o1 ≠ o2)
∧ ∃p.(has_dimension_value(o1,p,v)
        ∧ has_dimension_value(o2,p,v))
⇒ partial_containment(o1,o2)
```

The rule for *complementarity* is activated when two different observations have the same values for all of their shared dimensions and is summarized in the following:

```
observation(o1) ∧ observation(o2)
∧ (o1 ≠ o2)
∧ ∃p.(has_dimension_value(o1,p,v)
        ∧ has_dimension_value(o2,p,v))
∧ ∀p.(has_dimension_value(o1,p,v)
        ∧ has_dimension_value(o2,p,v))
⇒ complement(o1,o2)
```