Efficient Computation of Containment and Complementarity in RDF Data Cubes

Marios Meimaris
[1]University of Thessaly,
[2]ATHENA Research Center,
Greece
m.meimaris@imis.athena-innovation.gr

George Papastefanatos
ATHENA Research Center,
Greece
gpapas@imis.athena-innovation.gr

Panos Vassiliadis
University of Ioannina,
Greece
pvassil@cs.uoi.gr

Ioannis Anagnostopoulos
University of Thessaly,
Greece
janag@ucg.gr

ABSTRACT

Multidimensional data are published in the web of data under common directives, such as the Resource Description Framework (RDF). The increasing volume and diversity of these data pose the challenge of finding relations between them in a most efficient and accurate way, by taking into advantage their overlapping schemes. In this paper we define two types of relationships between multidimensional RDF data, and we propose algorithms for efficient and scalable computation of these relationships. Specifically, we define the notions of *containment* and *complementarity* between points in multidimensional dataspaces, as different aspects of relatedness, and we propose a baseline method for computing them, as well as two alternative methods that target speed and scalability. We provide an experimental evaluation over real-world and synthetic datasets and we compare our approach to a SPARQL-based and a rule-based alternative, which prove to be inefficient for increasing input sizes.

Categories and Subject Descriptors

H.2.8 [**Information Systems Applications**]: Database Management—*Database Applications*

Keywords

RDF, Multidimensional Data,OLAP, Information Extraction, Algorithms, Performance

1. INTRODUCTION

Over the past few years, a wide range of public and private bodies such as statistical authorities, academic institutions, financial organizations and pharmaceutical companies adopt RDF and the Linked Data (LD) paradigm [9, 31] to enable public access to multidimensional data in a variety of domains, including socio-economics, demographics, enterprise OLAP, clinical trials and health data. The published data enables third parties to combine information, perform analytics over different datasets and gain insights to assist data journalism, industry data management, and evidence-based policy making [25, 4, 27].

Multidimensional data are usually treated under the OLAP prism, where they are represented as *observations* that are instantiated over pre-defined *dimensions* and *measures* [7]. The dimensions provide context to the measures and are structured in hierarchies of different granularity *levels*. For example, a dataset that measures population broken down by locations and time periods, will consist of two dimensions, namely *location* and *time-period*, pertaining to granularity levels, such as countries, regions, and cities, or decades, years and quarters, respectively. A combination of fixed dimension levels is referred to as a *cube*; it contains the set of observations that instantiate these levels, such as the *populations of EU countries in the last decade*, or the *unemployment of EU cities in 2014*. All different combinations of dimension levels form a hierarchical cube *lattice*, where cubes are related with ancestry links.

In the context of Linked Data, the modelling recommendation is the Data Cube Vocabulary (QB) [9]. This provides a common meta-schema for mapping multidimensional data to RDF, enabling the representation of datasets, dataset schemas, dimensions, dimension hierarchies (e.g. codelists), measures, observations and slices (parts of datasets). An example observation can be seen in Listing 1, in which observation *obs1* measures the population of Germany in 2001. The values for 2001 (ex:Y2001) and Germany (ex:DE) are URIs from an example code list. The reuse of common URIs for handling multidimensional elements across different sources enables sharing of terms and the use of SPARQL for federating queries over remote datasets, laying the basis for the application of OLAP analytics at web scale.

Currently, there are few works on definitions and techniques for the discovery and classification of relationships between multidimensional observations in RDF settings [18, 32]. Such relationships can provide useful information to the analyst, such as whether an observation contains aggregated data with respect to other observations, and when two ob-

©2016, Copyright is with the authors. Published in Proc. 19th International Conference on Extending Database Technology (EDBT), March 15-18, 2016 - Bordeaux, France: ISBN 978-3-89318-070-7, on OpenProceedings.org. Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0
.

servations measuring different phenomena are complementary and can be combined. Assume that a data journalist is working on the issue of unemployment in different parts of the world and wants to explore whether unemployment rates relate to the population of a city or a country. Assume now that our journalist obtains dozens of datasets on the topic from various sources and wants to see whether and how they can be combined to facilitate his task. Although the journalist has defined his own reference dimension hierarchies[1] as shown in Figure 1, and converted the incoming data values to them via a simple script, it is still unclear to him that among the dozens of datasets he has collected (D_1, D_2 and D_3 of Figure 2), coming from different RDF sources, can be combined to support his task.

In our example, observation o_{11} shares the same dimension values with o_{31}, but they measure two different facts, which can be complemented. Furthermore, observations o_{21}, o_{22} that measure unemployment for 2011 in Greece and Italy, respectively, contain observations o_{32}, o_{33} that measure unemployment in Athens and Rome for a sub-period of the same year, although o_{21}, o_{22} measure poverty as well. The resulting table can be seen in Figure 3. This knowledge gives insights on how observations are related across datasets, how OLAP operations (roll-ups, drill-downs) can be applied in order to navigate and explore remote cubes, make observations comparable, provide recommendations for online browsing and quantify the degree of relatedness between data sources. Furthermore, materialization of these relationships helps speed up online exploration, and computation of k-dominance as defined in [6], and skylines or k-dominant skylines. Especially in the case of skylines, computation of containment between observations provides a means to directly access skyline, or k-dominant skyline points in collections of large web data.

However, placing different data into the same context and finding hidden knowledge is difficult and computationally challenging [4]. The detection of pair-wise relationships between observations is inherently a quadratic task; typically every observation from one dataset must be compared to all others within the same or from different datasets. The use of traditional ways such as SPARQL- or rule-based techniques becomes inefficient as the number of observations and the number of cubes increases. Our preliminary experiments employing recursive SPARQL queries with property paths and negation, indicate that even for 20k observations from 7 datasets it can take more than one hour to complete in commodity hardware, while the same tasks time-out for larger numbers of instances. The same holds when inference-based techniques are used: OWL-based reasoning lacks the expressivity for complex property-based inference, while rule-based reasoning such as SWRL [15] and Jena Rules [5] does not scale adequately due to the transitive nature of the relationships and the universal quantification needed to encode the conditions; the search space expands exponentially [12]. From the above, there is a clear need for establishing new efficient methods for computing pair-wise relationships between RDF observations that can scale up to the numbers and variety of datasets currently published on the web of data.

[1]In realistic settings, schema alignment is often necessary. Two prominent cases where it is used in practice include: (a) traditional BI environments, where all dimensions provide a reconciled *dimension bus*, and (b) user-initiated data collections from the web. In both cases, incoming data have to be translated to a reference vocabulary, *before* being used for further analysis.

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

Listing 1: Example RDF Data Cube observation

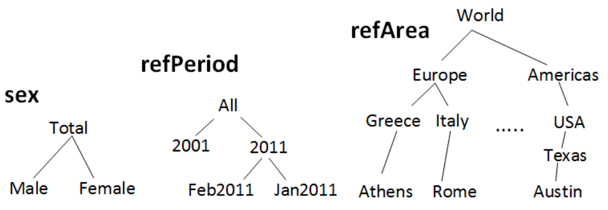


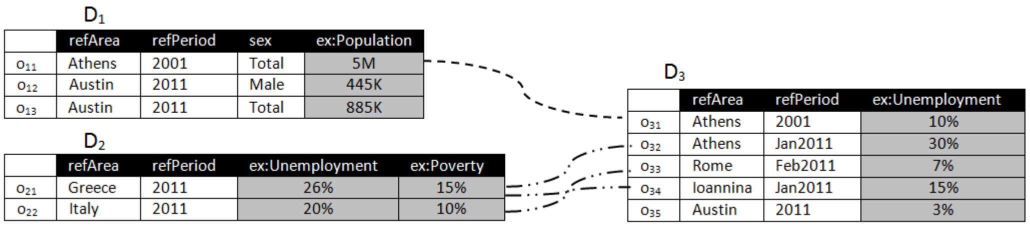
Figure 1: Hierarchical code list for the dimensions in Figure 2.

Approach Overview. In this paper, *we address the problem of efficiently computing containment and complementarity between RDF observations*, extending the preliminary work presented in [22]. Specifically, given an observation o_g, and a set of observations O from different sources, we define two relationships, namely *containment* and *complementarity.* A *containment* relationship captures whether an observation contains aggregated information with respect to other observations. It determines if values from the dimensions of o_g contain fully or partially values of the dimensions of another observation, thus enabling rollup and drilldown operations, directly or indirectly, as well as assigning them a hierarchy-based similarity metric. A complementarity relationship captures whether the measures of two observations can be combined together, providing comparable data for the same points in the multidimensional space. More specifically, we extend the notion of schema complement, defined in [10] and apply it at the instance level to discover complementary observations.

We first present a $O(n^2)$ baseline technique to calculate these properties. Then, we introduce two alternatives that prove to be more efficient and scalable in terms of computation time. The first performs observation *clustering* and limits comparisons between observations in the same clusters. The second uses a *multidimensional lattice* to assign observations to specific permutations of dimensions and levels, and prune the search space by checking for restrictions at the schema-level. We experimentally evaluate these 3 techniques over a set of 7 real-world multidimensional datasets and compare them in terms of efficiency with traditional techniques, namely with a SPARQL-based and a rule-based approach. Finally, we evaluate the scalability of our approach in an artificially generated dataset.

Contributions. The contributions of this paper can be summarized as follows:

- we extend the notions of *full* and *partial containment* between two observations defined in [22] as derivatives of the hierarchical relationships between their dimension values, and observation *complementarity* as a



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.

means of correlation between data points,

- we present a baseline, data-driven technique for computing these properties in memory,
- we introduce two alternative techniques that target efficiency,
- we evaluate our techniques in terms of efficiency and scalability over real-world and synthetic datasets, and we perform comparisons between them, as well as with a SPARQL-based and a rule-based approach.

The remainder of this paper is organized as follows. Section 2 presents preliminaries and formulates the problem of containment and complementarity computation between RDF cubes. Section 3 presents the baseline algorithm and two proposed improvements for computing the defined relationships. Section 4 describes the performed experiments and evaluation. Finally, section 5 discusses related work and section 6 concludes this paper.

2. PRELIMINARIES

The problem space is composed of n datasets, each of which contains a number of observations, structured in one or more cubes. A single dataset is composed of the schema part (i.e. dimensions, measures and attributes), and the data part (i.e. observations or facts). Dimension values are provided by code lists with hierarchical structure, whereas flat code lists pertain to dimensions with exactly one level.

Definition 1 (Dataset Structure): Let D $= \{D_1, \ldots, D_n\}$ be the set of all input datasets. A dataset $D_i \in D$ is composed by the set of observations, O_i, and the set of schema definitions, S_i, and $O = \{O_1, \ldots, O_n\}$ and $S = \{S_1, \ldots, S_n\}$ are the sets of all observations and schema definitions in D. Furthermore, a schema S_i consists of the sets of dimensions P_i and measures M_i defined in D_i, i.e., $S_i = \{P_i, M_i\}$. Let $P = \bigcup_{i=1}^{n} P_i = p_1, p_2, \ldots, p_k$ and $M = \bigcup_{i=1}^{n} M_i = m_1, m_2, \ldots, m_l$ be the set of all k distinct dimensions and l measure properties in D. Any $p_j \in P, m_j \in M$ can belong to more than one S_i, as dimension and measure properties are reused among sources. In our example, S_1, S_2 and S_3 are the schemata of datasets D_1, D_2 and D_3, and dimensions *refArea* and *refPeriod* belong to all three schemata. Similarly, measure *ex:unemployment* belongs to both M_2 and M_3. An observation $o \in O_i$ is an entity that instantiates all dimension and measure properties defined in S_i. The value that observation o_i has for dimension p_j is h_i^j. In the example, the values in the white cells represent dimension values (e.g. *"Athens"* is a value for dimension *refArea*), while grey cells represent measured values (e.g. *10% unemployment*).

Definition 2 (Hierarchies): Each dimension $p_j \in P$ takes values from a code list, i.e. a set of fixed values coded by URIs, $C(p_j) = \{c(p_j)_1, \ldots c(p_j)_m\}, j = 1 \ldots k$, (for simplicity we write c_{ji} instead of $c(p_j)_i$). Each code list defines a hierarchy such that when $c_{ji} \succ c_{jm}$, then c_{ji} is an ancestor of c_{jm}. Furthermore, we define c_{jroot} as the top concept in the code list of p_j , i.e., an ancestor of all other terms in the coded list, such that $\forall c_{ji} : c_{jroot} \succ c_{ji}$. This kind of ancestry is reflexive, i.e. $\forall c_{ji} : c_{ji} \succ c_{ji}$. In Figure 2, sample code lists are depicted for the three dimensions shown in Figure 1. In our example, *Greece* $\succ$ *Athens*, *Ioannina* and *Italy* $\succ$ *Rome*.

A *complementarity* relationship captures whether two observations measure different facts about the same set of dimension instances. In the motivating example of Figure 1, o_{11} and o_{31} are complementary because they measure different facts (population and unemployment respectively) about the city of Athens in 2001. 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. This is captured in the following definition.

Definition 3 (Observation Complement): Given two observations o_a and o_b and their dimensions P_a and P_b, o_a complements o_b when the following conditions hold:

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

$$\forall p_j \in P_b \setminus P_a : h_b^j = c_{jroot} \tag{2}$$

Therefore, (1) $\wedge$ (2) $\Rightarrow Compl(o_a, o_b)$. We denote this with $Compl(o_a, o_b)$. Definition 1 states that the shared dimensions must have the same values as stated in (1), and all other dimension values of o_b must be equal to the root of the dimension hierarchy, providing no further specialization, as stated in (2). For example, an observation measuring poverty in Greece is observation complement with an observation measuring the population in Greece for all human genders. 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 c_{jroot} (i.e. no specialization).

A *containment* relationship captures whether an observation aggregates the measures of the contained observations. For example, measuring the population of Greece implicitly contains all the populations of Greece's sub-regions. We distinguish between *full* and *partial* containment. The former denotes that all contained observations must be aggregated (e.g., a roll-up operation) for being observation complement with the containing one, while the latter denotes that both contained and containing observation must be rolled-up on their disjoint dimensions for becoming complementary. These two concepts are defined as follows.

Definition 4 (Partial and full containment): Given two observations o_a and o_b, their dimensions P_a and P_b and their measures M_a and M_b, partial containment between o_a and o_b exists when the following conditions hold:

$$M_a \cap M_b \neq \emptyset \tag{3}$$

$$\exists p_i \in P_a \cap P_b : h_a^i \succ h_b^i \tag{4}$$

Therefore, (3) $\wedge$ (4) $\Rightarrow Cont_{partial}(o_a, o_b)$. An observation o_a partly contains o_b when there is at least one M_i shared between o_a and o_b as stated in (3), and 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 (4). We denote this as $Cont_{partial}(o_a, o_b)$. In the example, observation o_{21} partially contains o_{31}, because *Greece* contains *Athens* but *2001* does not contain *2011*. By rolling up on the *refPeriod* dimension, the two observations become complementary.

Similarly, full containment between o_a and o_b exists when the same preconditions (3)-(4) hold along with a universal restriction on (4) *for all* dimension values, i.e:

$$\forall p_i \in P_a \cap P_b : h_a^i \succ h_b^i \tag{5}$$

Therefore, (3) $\wedge$ (4) $\wedge$ (5) $\Rightarrow Cont_{full}(o_a, o_b)$. An observation o_a fully contains o_b when there is one M_i shared between o_a and o_b as stated in (3), and values of all dimensions for o_a are hierarchical ancestors of the values for the same dimensions in o_b as stated existentially in (4) and universally in (5). We denote this with $Cont_{full}(o_a, o_b)$. Observe that the containment property is not symmetric and that given $Cont_{full}(o_a, o_b)$, then $Cont_{full}(o_b, o_a)$ is not implied. In the example, o_{21} fully contains o_{32} and o_{34}. The notation is summarized in Table 1. Based on the above, our problem is formulated as follows.

Problem: Given a set D of source dataset, 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) $Cont_{full}(o_i, o_j)$, b) $Cont_{partial}(o_i, o_j)$ and c) $Compl(o_i, o_j)$. 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
P_i	The set of dimensions for observation o_i
p_i	The i-th dimension in a set P_k
M_i	The set of measures for observation o_i
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
c_{jroot}	The root value (ALL) for dimension p_j
$Compl(o_a, o_b)$	Observation o_a complements o_b
$Cont_{full}(o_a, o_b)$	Observation o_a fully contains o_b
$Cont_{partial}(o_a, o_b)$	Observation o_a partially contains o_b

3. ALGORITHMS

In this section, we present three approaches for computing containment and complementarity relationships. We first present a baseline method of $O(n^2)$ complexity and then we propose two alternatives in order to achieve scalable and fast solutions.

3.1 Baseline

Our *baseline* algorithm performs pair-wise computations between all pairs of observations in a given data space. To achieve this, we model observations as bit vectors in a multidimensional data space represented by an occurrence matrix **OM**, where rows represent observations and columns are values in their dimensions as well as ancestor values in their dimension hierarchies. **OM** encodes the occurrence of a dimension value in an observation as well as the hierarchy in which this value belongs to, by setting the value of 1 to all columns that are ancestors of this value.

Dimension alignment is often required to take place before this step, in order to have a reconciled dimension bus in the feature space. As discussed later on in the experiments section, we employ a state-of-the-art tool for performing the interlinking of dimension values across different datasets. Note that this procedure is orthogonal to the work presented in this paper. The problem of dimension alignment, and record linkage in general, is a separate research problem and is not underestimated; however, (a) data conversion is feasible and amortized over time (esp., if data collection from "favorite" sources is recurring), and (b) the focus of this paper is on analytics and not data integration.

Constructing the Occurrence Matrix. Each observation o_i defines a bit vector $\mathbf{o_i}$ and all $\mathbf{o_i} \in O$ comprise the occurrence matrix **OM** of $|O| \times |C|$ dimensions, defined by the occurrences of code list values in the respective dimensions. Each value $c_{ji} \in C_j$ corresponding to dimension p_j is a feature, i.e., a column in **OM**. Hierarchical containment is encoded into **OM** using a bottom-up algorithm that assigns a value of 1 in column c_{ji} and all of its parents, if the value h_a^j of the dimension p_j of o_a is equal to the feature c_{ji}. Finally, we set the c_{jroot} columns of all observations that do not contain p_j in their schema. This means that dimensions not appearing in a schema are mapped to the *top* concept (i.e. root term), including all possible values.

Matrix **OM** can be further broken down in separate submatrices for each code list, i.e., $\mathbf{OM} = [\mathbf{OM_1}, \ldots, \mathbf{OM_{|C|}}]$ for all dimensions, where $\mathbf{OM}_i$ is a sub-matrix that represents occurrences for all values of dimension p_i. Matrix **OM** for the example of Figure 2, given the hierarchies shown in Figure 1, is shown in Table 2. **OM** is used as an input for our algorithm that computes a containment matrix. The latter is used for assessing both complementarity and containment properties.

Computation of containment. A containment matrix $\mathbf{CM_i}$ captures pair-wise containment between observations, for dimension p_i. If a cell $\mathbf{CM_i}[a, b] > 0$, then o_a and o_b are related via containment for p_i. To compute $\mathbf{CM_i}$, we apply a conditional function between pairs of rows, as bit vectors. Containment exists if the logical AND operation between the bit vectors of the rows returns one of the two bit vectors. We define this check as the following conditional function applied on o_a and o_b:

$$sf(o_a, o_b) \mid_{(p_i)} = \begin{cases} 1, & \text{if } a \wedge b = b \\ 0, & \text{otherwise} \end{cases}$$

Notice that 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, $\mathbf{CM_1}, \ldots, \mathbf{CM_k}$. By adding these we get the *Overall Containment Matrix* $\mathbf{OCM}$:

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

$\mathbf{OCM}$ values are normalized; we derive *full containment* 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 $\mathbf{OCM}$ matrix is explained in Algorithm 1 *computeOCM*. We then calculate containment and complementarity using the $\mathbf{OCM}$-based Algorithm 2 *baseline*.

Computation of complementarity. Following the definition of observation complementarity, and given that the non-appearing values are set to c_{jroot}, we use $\mathbf{OCM}$ to assess whether a pair of observations exhibits full containment in both directions, i.e. $Cont_{full}(o_a, o_b)$ and at the same time $Cont_{full}(o_b, o_a)$.

Analysis. The baseline algorithm has $\Theta(n^2)$ time complexity for n observations, because it visits each pair of observations exactly once, if all three types of relationships are to be retrieved. The occurrence matrix $\mathbf{OM}$ requires $\Theta(nk)$ space for n observations and k features, given a simple array implementation. However, for large k the matrix tends to become sparse, therefore a sparse matrix implementation would yield a significant decrease in the required space. In practice, if at least one 0 is found in the $\mathbf{CM}$ matrices, the pair under comparison is no longer candidate for either full containment or complementarity. We keep this in an index table to avoid meaningless comparisons in the next step, if only full containment or complementarity is to be computed. Finally, if we are not interested in identifying the particular dimensions that exhibit containment in a partial containment relationship, we skip iterating through the $\mathbf{CM_i}$ matrices and identifying the zero values, and just keep the value as a metric of the degree of partial containment that the pair exhibits.

3.2 Computation with Clustering

The baseline algorithm becomes inefficient for large datasets and does not scale due to its quadratic complexity. For this reason, we improve its performance by applying a preprocessing step that prunes the search space by limiting comparisons. Algorithm 3 presents our approach. It takes as input the occurrence matrix containing the bit vectors for the observations and applies a clustering algorithm that splits the matrix into smaller occurrence matrices. It then applies the *computeOCM* and *baseline* algorithms to each separate cluster. This way comparisons between pairs of observations are limited within each cluster. This is shown in Algorithm 3.

Algorithm 1 *computeOCM*

Input: An occurrence matrix **OM** with N rows and $|C|$ columns, a set P of dimensions and their start indices in **OM**
Output: A NxN overall containment matrix **OCM**

```
1: initialize $\mathbf{OCM}[][]$
2: for each $p_i \in P$ do
3:     initialize $\mathbf{CM_{pi}}[][]$    ▷ one containment matrix per dimension
4:     for each $o_j \in \mathbf{OM_{pi}}$ do    ▷ $p_i$ defines a start index
5:         for each $o_k \in \mathbf{OM_{pi}}$ do
6:             if $o_j$ AND $o_k == o_j$ then
7:                 $\mathbf{CM_{pi}}[j][k] \leftarrow 1$
8:             else
9:                 $\mathbf{CM_{pi}}[j][k] \leftarrow 0$
10:            $\mathbf{OCM}[j][k] \leftarrow \mathbf{OCM}[j][k] + (CM_(p_i)/|P|)$ ▷ normalize for # of dimensions
   return OCM
```

Algorithm 2 *baseline*

Input: A NxN overall containment matrix **OCM**.
Output: S_F, S_p, S_c sets of fully, partial containment and complementarity relationships, and optionally 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 $o_i \in \mathbf{OCM}$ do
3:     for each $o_j \in \mathbf{OCM}$ && $o_j \neq o_i$ do
4:         if $\mathbf{OCM}[i][j] == 1$ then
5:             $S_F = S_F \cup (o_i, o_j)$
6:             if $\mathbf{OCM}[j][i] == 1$ then
7:                 $S_C = S_C \cup (o_i, o_j)$
8:         else if $\mathbf{OCM}[i][j] > 0$ then
9:             $S_P = S_P \cup (o_i, o_j)$
10:            for each $p_i \in P$ do
11:                if $\mathbf{CM_{pi}}[i][j] == 1$ then
12:                    $map_P(o_i, o_j, p_i) = true$
13:        else continue
   return $S_F, S_P, S_C, map_P$
```

Clustering configuration. In order to compute clusters, state-of-the-art clustering algorithms can be employed. We have experimented with k/x-means [26], bottom-up hierarchical clustering and fast canopy clustering [21] for the purposes of evaluating our algorithms. To avoid introducing extra time overhead in the containment and complementarity computation stage, we approximate the algorithm by clustering a sample of the data and assigning the remaining points to the identified clusters.

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 dis-

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

Algorithm 3 *clustering*

Input: An occurrence matrix **OM** with N rows and $|C|$ columns.
Output: S_F, S_p, S_c sets for fully, partial containment and complementarity relationships.

```
1: $clusters[] \leftarrow cluster(\mathbf{OM}, \dots)$          ▷ e.g. k-means
2: initialize $\mathbf{OCM} \leftarrow 0$
3: for $i = 1$ to clusters.$size$ do
4:     $\mathbf{OCM_i} \leftarrow$ computeOCM(clusters[$i$], $P$)
5:     $S_{Fi}, S_{Pi}, S_{Ci} \leftarrow$ baseline($\mathbf{OCM_i}$)
6:     $S_F, S_P, S_C \leftarrow (S_F, S_P S_C) \cup (S_{Fi}, S_{Pi}, S_{Ci})$
   return $S_F, S_P, S_C$
```

tribution 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.

3.3 Computation with Cube Masking

The efficiency achieved by the clustering approach can result in lower recall levels as observations that are likely to be related might end up in different clusters. In this section, we propose a scalable algorithm that greatly increases speed and at the same time maintains 100% recall. The approach is based on the hierarchical characteristics of the dimension values. We first construct a lattice of all possible level combinations for all the dimension values that appear at least once in the input. Then, we map all observations to their respective lattice nodes (i.e. cubes) and check if the lattice nodes, rather than the observations, meet the containment and complementarity criteria. If so, comparisons need only be performed between observations belonging to these lattice nodes. To demonstrate this further, consider the lattice shown in Figure 3 that corresponds to the dimension hierarchies of Figure 2. Each node corresponds to a combination of the levels of all dimensions. Therefore, node ”210” represents the cube of all observations that pertain to values at level 2 for *refArea*, level 1 for *refPeriod* and level 0 for *sex*. 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 dimension inclusion (i.e. path) in the lattice before comparing the contents.

Based on these observations, we propose the *cubeMasking* algorithm (Algorithm 4) that works in the following steps: i) It first identifies cubes in the input datasets and populate the lattice; ii) it maps observations to cubes; iii) it iterates through cubes and does a pair-wise check for the containment criterion and finally iv) it compares observations between pairs of cubes that fulfils this criterion. 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 4.

Analysis. Instance-level comparisons are limited between pairs of comparable cubes that are identified by the algorithm. At worst, the maximum number of cubes for a set of input datasets is the number of permutations of dimensions and levels, i.e. $k^{(|P|)}$, where k is the maximum level of all hierarchies and $|P|$ is the number of dimensions. Checking for potentially comparable pairs of cubes costs $O(k^{(2|P|)})$ in the worst case, but for an average of $bk^{(|P|)}$ present cubes in the input, and $abk^{(|P|)}$ number of comparable cubes, where a and b are constants between 0 and 1, the algorithm will require ab^2n^2 comparisons instead of n^2.

4. PERFORMANCE EVALUATION

Datasets. We have experimented with seven real-world datasets taken from the statistics domain. Eurostat Linked

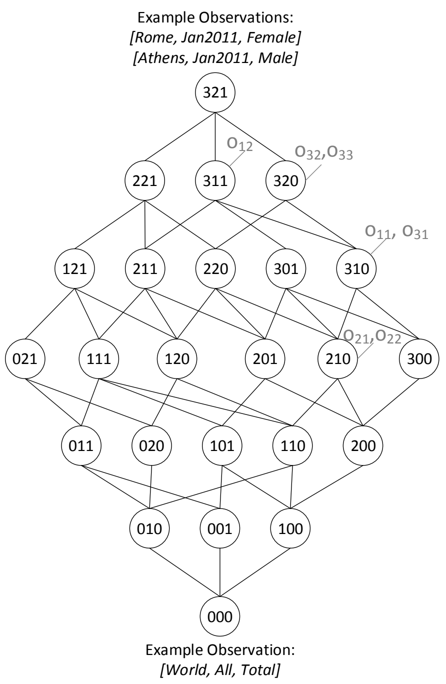


Figure 4: The lattice for the three hierarchies of Figure 2. Observations in Figure 1 are mapped to the appropriate node. The number in each node corresponds to the level of each dimension.

Data Wrapper, the Eurostat database, linked-statistics.gr and World Bank[2] were used as the dataset sources. The datasets were either in RDF form, or converted to RDF. In the case of datasets in CSV format, we follow the approach of [28] for producing RDF QB datasets, although many other popular tools can be used, such as CSV2RDF[3], OpenCube[4] and Open Refine[5]. We converted CSV column headers to dimension URIs, and rows to observations, by automatically matching cell values to existing code list terms based on their IDs. The datasets contain information of European authorities on unemployment, resident population, number of households, births, deaths and gross domestic product (GDP) on a multitude of dimensions that include locations, dates, citizenship, sex, household size, education level and economic activity. In total, the datasets amount to 250k observations and 2.6k distinct hierarchical values. These exhibit an overlap of 11 dimensions pertaining to common coded lists, and 6 measures.

In our experiments, we consider a separate preprocessing step for the alignment of the schemas and the mapping of the dimension values across the input datasets. In this work we have used LIMES [24], a state-of-the-art link discovery framework commonly used for entity matching tasks in the Web of Data. LIMES is configurable to use SPARQL query restrictions on input data (e.g. *only match nodes of type skos:Concept*), and has a rule language that enables

[2]`http://estatwrap.ontologycentral.com/` , `http://data.worldbank.org/` , `http://linked-statistics.gr/` , `http://epp.eurostat.ec.europa.eu/portal/page/portal/statistics/search_database`
[3]`http://www.w3.org/TR/csv2rdf/`
[4]`http://opencube-toolkit.eu/`
[5]`http://refine.deri.ie/`

Algorithm 4 *cubeMasking*

Input: 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 of N observations with their descriptions.
Output: S_F, S_p, S_c sets for full, partial containment and complementarity relationships.

```
 1: $hierarchy \leftarrow createHierarchyTree(C)$   ▷ iterate through observations once to identify cubes and map observations to cubes
 2: initialize $cubeLevels$, $observationsInCubes$
 3: for each $o_i \in O$ do
 4:     initialize $cube$
 5:     for each $p_j \in P$ do
 6:         $cube.p_j \leftarrow$ levels$(o_i, p_j)$
 7:     $cubeLevels.add(cube)$   ▷ hashing ensures no duplicates
 8:     $observationsInCubes(o_i) \leftarrow$ cube
 9: for each $cube_j$, $cube_k \in cubeLevels$ do
10:     for each $p_i \in P$ do
11:         if $not(cube_j.p_i \prec cube_k.p_i)$ then break
12:         for each $o_i \in cube_j$ do
13:             for each $o_j \in cube_k$ do
14:                 $\mathbf{SF}[o_i, o_j] \leftarrow checkFullCont(o_i, o_j)$
15:                 $\mathbf{SP}[o_i, o_j] \leftarrow checkPartialCont(o_i, o_j)$
16:                 $\mathbf{SC}[o_i, o_j] \leftarrow checkCompl(o_i, o_j)$
    return $S_F, S_P, S_C$
17: procedure CREATEHIERARCHYTREE($C$)
18:     initialize $hierarchyTree$
19:     for each $c_i \in C$ do
20:         $hierarchyTree.add(c_i)$
21:         for each $c_i.parent$ do
22:             $hierarchyTree.addParent(c_i, c_i.parent)$
        return $hierarchyTree$
23: procedure CHECKFULLCONT($o_i, o_j$)
24:     for each $p_i \in P$ do
25:         if not $hierarchy.isParent(o_i.p_i, o_j.p_i)$ then return false
        return true
26: procedure CHECKPARTIALCONT($o_i, o_j$)
27:     for each $p_i \in P$ do
28:         if $hierarchy.isParent(o_i.p_i, o_j.p_i)$ then return true
        return false
29: procedure CHECKCOMPL($o_i, o_j$)
30:     if $checkFullCont(o_i.p_i, o_j.p_i)$ && $checkFullCont(o_j.p_i, o_i.p_i)$ then return true
        return false
```

the user to select combinations of distance functions (e.g. the maximum of the *cosine* and *levenshtein* distances). We configured LIMES to match hierarchy nodes by adding their URIs as literal values, and used their cosine distance in order to find close matches based on the identifiers usually found in the suffix part of a URI. The details for each dataset are summarized in Table 4.

Metrics. The goal of the experiments was to assess and compare the performance of the proposed algorithms with respect to *execution time* and *recall* of computed relationships. *Execution time* is measured as the time needed for data pre-processing and for computing complementarity and containment properties. *Recall* is calculated as the ratio of computed relationships to actual relationships. However,

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

it is relevant only to the *clustering* algorithm, as the *baseline* and the *cubeMasking* algorithms achieve 100% recall. Note, also, that the precision is 100% for all algorithms as a derivative of the determinism in the relationship definitions. In order to derive recall, we have compared the output of the *clustering* method with the ground truth taken from the *baseline* output. Note that we do not consider any decrease in recall induced by the tool used in the dimension alignment step. Since the recall of our approach is not dependent to the result of the alignment process, we assume that the output of the linking tool (in our case LIMES) achieves 100% recall correctly matching schema elements and dimension values across all datasets.

Experimental Setting. Our approach was implemented in Java 1.7, and all experiments were performed on a server with Intel i7 3820 3.6GHz, running Debian with kernel version 3.2.0 and allocated memory of 16GB. We implemented the approaches and performed a series of comparisons between them, starting from an input size of 2k observations, over the original fixed size of dimensions. We continued the experiment with an input size of 20k and then we further increased it with a 20k step. For the *clustering* method, we have experimented with canopy clustering, hierarchical clustering and x-means, all applied on a 10% random sample of the original datasets and empirical studies showed that x-means outperformed the other two methods greatly in terms of recall achieved in comparable time frames. Based on the bit-vector approach, we used the Jaccard coefficient as a similarity metric for our binary feature space. The *cubeMasking* algorithm has been further optimized to limit the number of cube comparisons by storing for each cube, the full set of its children in memory.

SPARQL-based. We have experimented with SPARQL queries over a Virtuoso 7.1 instance for computing the relationships. Note that universal quantification as well as recursive querying (i.e. property paths) are necessary to compute full containment and complementarity. Property paths are directly supported by SPARQL 1.1, however, universal quantification must be mimicked by using a negation construct that includes a nested recursion, which makes the queries costly. Furthermore, occurrence of partial containment can be detected by SPARQL queries easily, but it is complicated and costly to quantify it. The SPARQL approach is composed of three SPARQL queries; for simplicity, we are only interested in detecting the underlying *existence* of the containment and complementarity relationships, and we do not quantify it like in the computation of the OCM matrix. In the case of *partial containment*, the query for detecting pairs of observations is as follows:

```
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 privie the number of dimensions that participate in the *partial containment*; this would make the query more complicated. In the case of *complementarity*, we tested the data against the following SPARQL query:

```
SELECT ?o1, ?o2
WHERE {
        ?o1 a qb:Observation .
        ?o2 a qb:Observation .
        FILTER NOT EXISTS {
                ?o1 ?d1 ?v1.
                ?o2 ?d1 ?v2.
                FILTER(?v1!=?v2)
        }
        FILTER(?o1 != ?o2)
}
```

The query will be matched against pairs of observations whose shared dimensions do not have different values. In both queries, we have relaxed the conditions presented in section 2 regarding the observations' schema.

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 expressivity. The rule for computing *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)
```

In essence, we are checking for pairs of different observations that exhibit both existential and universal quantification in having dimension values subsume each other. The existential quantification is needed to ensure that there exists at least one such relationship, while the universal is needed to ensure that all relationships hold true. 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)
```

All datasets along with the experiment code are available online at `http://github.com/mmeimaris` .

4.1 Experimental Results

Our experiments indicate that the quadratic *baseline* algorithm is improved substantially by our two proposed alternative methods, i.e. *clustering* and *cubeMasking*. Specifically, we have achieved improvement by roughly one order of magnitude for the computation of *full containment* and *complementarity* by using the *cubeMasking* algorithm, while we find that the *clustering* algorithm exhibits a trade-off between execution time and relationship recall. The results can be seen in Figure 5.

Baseline. The *baseline* behaves as expected, performing n^2 comparisons for n observations. The partial containment relationships are quantified with a value between 0 and 1, non-inclusive, which increases monotonically in proportion with the number of dimensions that a pair of observations exhibits containment in. The results for computing the relationships can be seen in Figure 5(a-c). However, as can be seen in the Figure, it is cheaper to compute only full containment and complementarity because the cases where these do not apply can quickly be ruled out in the *computeOCM* step. This approach does not scale with respect to input size, as all pairs of rows in the OCM matrix, representing observations, have to be visited in order to detect and quantify containment and complementarity. It should be noted that, when having computed full containment, complementarity can be detected a posteriori by iterating through the fully contained pairs and checking for mutual (i.e. bi-directional) full containment.

Clustering. In the case of *clustering*, we have experimented with three different clustering algorithms, one agglomerative and two centroid-based, namely *hierarchical*, *canopy* and *x-means* respectively. Figure 5(d) shows the recall levels of the three algorithms. We have found that x-means, even when applied to a random 10% sample of the data, outperforms the other two in the resulting recall. Overall, the clustering method shows promising results and leaves room for improvement as far as the clustering step is concerned. Note that by definition, the *baseline* algorithm is equivalent to *clustering* with exactly one cluster. A large number of clusters k will limit the total number of comparisons, and consequently make the computations faster, but the process will be lossy, ultimately achieving lower recall levels.

Cube Masking. The *cubeMasking* algorithm is the fastest of all the approaches we experimented with, mainly because of the linear cost of identifying and assigning observations to cubes, and the reduced number of performed comparisons. This is a consequence of the fact that it takes advantage of the distributions of dimension levels and values, and reduces the needed comparisons to a minimum with respect to the baseline and clustering algorithms, while maintaining full recall. Furthermore, the experiments on synthetic data, as well as the observation that the ratio of cubes per input size tends to decrease as input size increases as depicted in Figure 5(f), show that it is scalable for big datasets. In essence, Figure 5(f) shows that the number of cubes in a collection of datasets will increase in a lower rate than the number of input observations. This implies that as the input size increases, the *cubeMasking* algorithm will usually not result in intractable pair-wise comparisons.

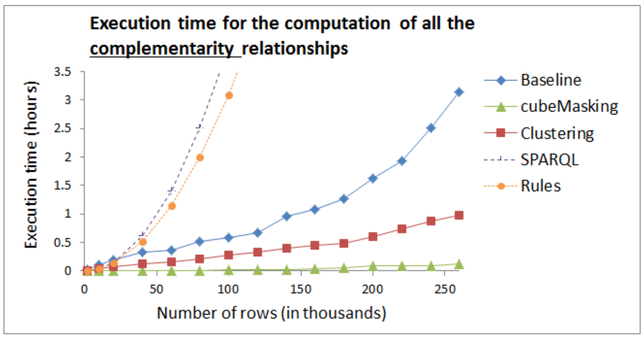
With cubeMasking, we need to check for parent-child relationships between cubes. The brute-force way to achieve this is to iterate through all pairs of cubes and check for pair-wise ancestry. This ancestry exists when all dimension levels of the first cube are equal or less than their respective dimension levels of the second cube. By pre-fetching and storing all children of each cube in memory, we introduce a conditional optimization that yields roughly 15-20% faster execution time for any input size as can be seen in Figure 5(g). However, creating this mapping implies either an explicit iteration of all cubes, which is costly, or an unavoidable iteration for one of the relationship types, which can be taken advantage of for the other two. Furthermore, keeping a list with the children of any given cube node adds an extra memory overhead, however in this work we are not interested in the memory footprint of each method.

SPARQL and Rule-based. These approaches perform adequately for small inputs, as shown in Figure 5(a), (b) and (c), but either hit the time-out limits or consume all memory resources quickly, which renders them unusable for the computations of such relationships over real world datasets. This is mainly due to the fact that the multi-transitive nature of the containment relationships creates an intractable search space, which is compensated by the dedicated specificities of our approach. From a reasoning-based perspective, it has been argued in the literature that dedicated reasoners tend to out-perform general purpose ones [8]; a fact that is supported by our experiments.

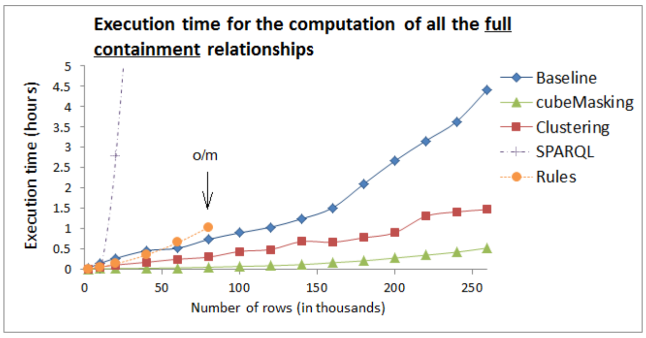
4.2 Scalability

In order to test scalability of our methods, we have created a synthetic dataset of 2.5M observations, following a similar approach as in [11]; the creation of synthetic data was based on fixing the number of dimensions and creating observations that follow a projected distribution of the data w.r.t to the real-world datasets. More specifically, we calculated the projected number of lattice nodes to reflect Figure 4(f), where the change in active lattice nodes is shown as the input size increases. Then, we populated the lattice nodes evenly.

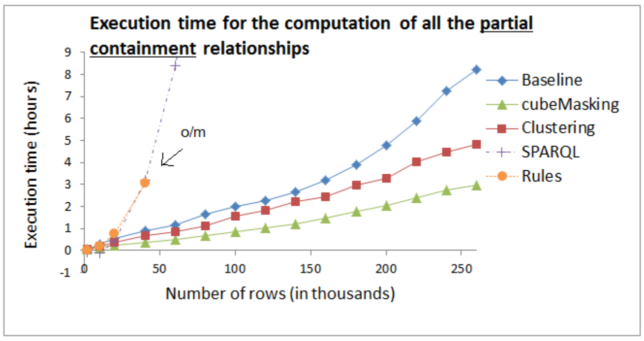
Experiments in the datasets show that the approaches utilizing SPARQL querying and rule inferencing do not scale adequately, even for small input sizes. Figure 5(e) shows how the three proposed methods scale in respect to the in-



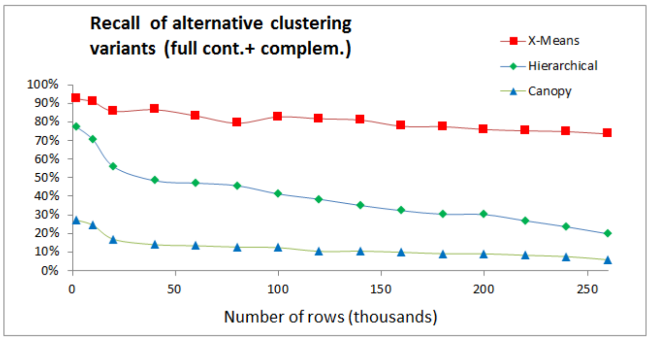
(a) Execution time (hours) for complementarity



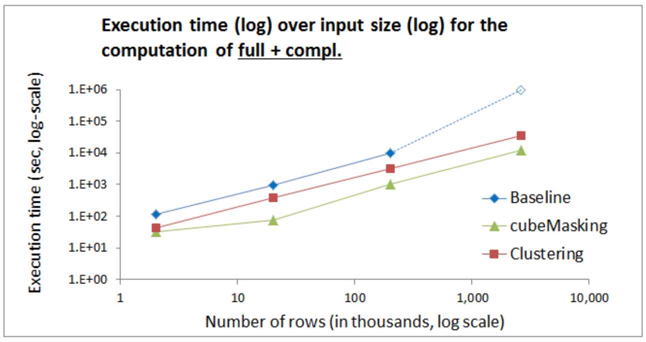
(b) Execution time (hours) for full containment



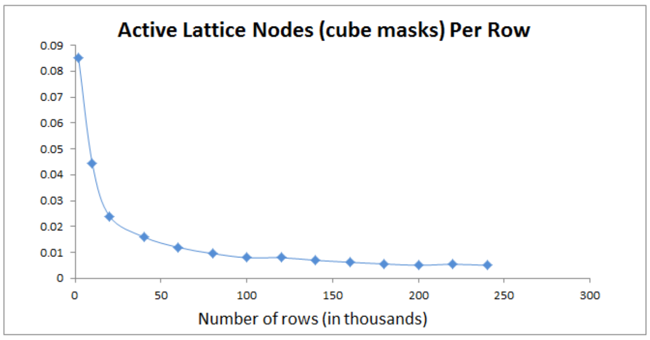
(c) Execution time (hours) for partial containment



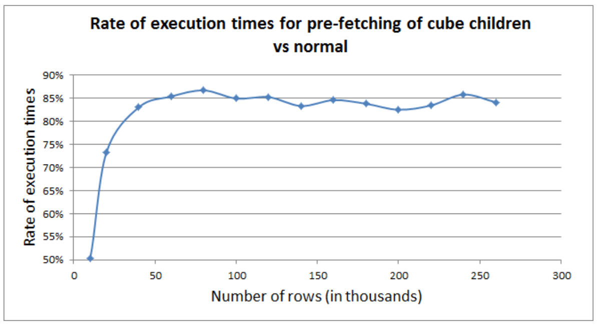
(d) Recall for three clustering algorithms w.r.t. input size



(e) log-log time and input size



(f) ratio of discovered cubes per observation count



(g) Rate of execution time with children pre-fetching vs normal for full containment

Figure 5: (a) Execution time (hours) for complementarity, (b) full containment, (c) partial containment (note that in the SPARQL based approach, partial containment is only detected and not quantified), (d) recall for three clustering algorithms w.r.t. input size, (e) log-log time and input size, (f) ratio of discovered cubes per observation count. o/m=out of memory, (g) rate of execution time with children pre-fetching vs normal for full containment.

put size. While the values for *clustering* and *cubeMasking* are based on measured values on the synthetic data, the value for the *baseline* method for 2.5m records is a projection of the quadratic analysis; it took more than 7 days to complete.

The results indicate that both *clustering* and *cubeMasking* scale better than the baseline, with the latter having a more clear advantage because of the reduced number of needed comparisons in conjunction with the fact that it is lossless. However, in extreme cases where the number of cubes is large and the distribution of observations in these cubes is sparse, the *cubeMasking* method will resemble the baseline. In these cases, the *clustering* method yields a more realistic advantage, especially when efficiency is more important than recall.

5. RELATED WORK

In this paper, we extend the work presented in [22], where we introduce preliminary notions of containment and complementarity, we outline a method of computing these notions, and define an RDF vocabulary as an extension of RDF QB for containment and complementarity between multidimensional observations across sources.

The problem of finding related observations in multidimensional spaces has been addressed within several contexts. Online Analytical Mining (OLAM) refers to the application of data mining in OLAP, and addresses OLAP classification [20], outlier detection[1], intelligent querying [28] and recommendation for OLAP exploration, which is based on either query formulation or instance similarity [2, 3]. These approaches enable discovery of latent information, exploratory analysis [13] and efficient querying [20].

In [14] the authors discuss how queries containing grouping and aggregation, which in our case is similar to observation containment, can be facilitated by materialized views. Partial containment is referred to as k-dominance in [6], where it is used to define partial skylines in multidimensional datasets.

Skyline computation in multidimensional datasets uses the notion of dominance, or observation containment, in order to assert whether a point is part of a dataset's skyline [33, 30, 19]. Skyline points are essentially top-level observations, i.e. observations that are not contained by other observations. However, these approaches are concerned with only skyline data points, rather than computation of all containment relationships. Skyline computation is, however, a direct derivative of containment computation.

Aligon et al.[2] address similarities between OLAP sessions by defining distance functions over query features. They experiment with Levenshtein, Dice's coefficient, tf-idf and Smith-Waterman, with the latter being the best for their purposes, whereas in our approach the Jaccard Coefficient addresses the binary nature of our feature space directly. Baikousi et al.[3] define distance functions for dimension hierarchies. However, they address observation similarity in general, rather than computing strict relationships as in our case. Hsu et al. [16] apply multidimensional scaling methods and hierarchical clustering in a hybrid approach in order to measure similarity between reports in the same cubes.

Business model ontologies have been deployed in the context of OLAP cubes in [29], where the authors define notions such as *merge* and *abstraction* of cubes. The *abstraction* notion is of particular interest to our work, as it resembles the containment relationship, however the setting and motivation behind this work is representational rather than computational.

In the context of RDF, finding related cubes in multidimensional contexts is addressed in [18], where the work is focused in extending the *Drill-Across* operator to address different sources. The authors define conversion and merging correspondences between remote cubes in order to quantify the degree of their overlap and enable meaningful combinations of datasets. However, they do not address specific relationships at the instance level. The need for RDF-specific workflows is addressed in [17], where the authors argue that analytical processing of RDF cubes requires more than the capabilities offered by SPARQL engines for querying, exploration and analysis, and are best complemented with OLAP-to-SPARQL engines that use RDF aggregate views and partial materialization. This is in favour of our approach that tackles efficient materialization of batch relationships.

The more general problem of finding similarity between resources is a main component in entity resolution, record linkage and interlinking [23, 24, 32]. These approaches deal with discovering links between nodes from different datasets by using distance-based techniques. To the best of our knowledge, this is the first work that addresses the definition, representation and computation of relationships between individual multidimensional observations.

6. CONCLUSIONS AND FUTURE STEPS

In this paper, we have presented and compared three novel approaches for discovering relationships between observations of multidimensional RDF data. We have defined new relationships, namely full and partial containment, and complementarity between observations as derivatives of the hierarchical relationships between their dimension values, and as a means of comparison and correlation of their measures. We performed an experimental evaluation and comparison between them and with a SPARQL-based and a rule-based technique and we show that our methods outperform the traditional approaches in both execution time and scalability. As this work is based on batch analysis, in the future we plan to study and define efficient incremental techniques, as well as hybrid probabilistic methods that take into advantage the positive points of the *clustering* and *cubeMasking* algorithms. We will also address space efficiency and examine the behaviour of our approaches on settings with memory restrictions. Finally, we intend to examine the performance of our algorithms in distributed and parallel contexts.

Acknowledgements. This work is supported by the EU-funded ICT project "DIACHRON" (agreement no 601043).

7. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] S. Coppens, M. Vander Sande, R. Verborgh, E. Mannens, and R. Van de Walle. Reasoning over sparql. In *LDOW*. Citeseer, 2013.

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

[10] 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.

[11] 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.

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

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

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

[29] C. Schütz, B. Neumayr, and M. Schrefl. Business model ontologies in olap cubes. In *Advanced Information Systems Engineering*, pages 514–529. Springer, 2013.

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

[31] 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.

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

[33] 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.