Contents

1. Linked Data

Linked Data is a set of best practices for publishing and connecting data on the Web structured in such a way that it is usable not only for human processing but also processable by machines. It builds upon the general architecture of the World Wide Web. Instead of creating links between particular documents from different sources in the case of the classic Web of documents, the Linked Data connects the representations of real-world objects or abstract concepts. Tim Berners-Lee expressed these best practises in four principles, known as the Linked Data principles.[2]

- Use URIs as names for things.
- Use HTTP URIs, so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards
- Include links to other URIs, so that they can discover more things.

URIs (Universal Resource Identifier) are used to identify real-world objects and abstract concepts. According to the second principle, information about the entity that the URI represents can be retrieved using HTTP protocol (so-called URI dereferencing). Based on the third principle, which advocates for a standard structure of data dereferenceable by URI, the Resource Description Framework (RDF) has been designed. Tim Berners-Lee also suggested 5-star deployment scheme for ranking published data according to the format in which it is published, comparing them based on their ability to be machine processed. It assigns one star to any data published and assigns the highest number of five stars to data, that is published as RDF, where the entities described are identified by an dereferenceable URI string and the data are connected to other data sources.

1.1 Resource Description Framework

RDF is a data model based on representing data as directed graphs. The basic building block of RDF structured data is a triple consisting of three parts called subject, predicate, and object. The subject is the URI representing the described resource. The object is either URI or literal value like string or number. The predicate specifies the type of relationship between the resources at the positions of subject and object. The predicate is always identified by URI. Predicate URIs come from vocabularies, intended to encompass various relations and concepts occurring in a certain domain.

Set of triples then establishes a RDF graph. URIs at the subject and object positions of the triples make nodes of the graph and each triple acts as an arc connecting the nodes. Type of the connecting is expressed by the predicate URI in the triple. Given the uniqueness of the URIs and their capability of being dereferenced and connected to URIs from various sources (the fourth Linked Data principle), one can imagine the linked data as one giant undivided

graph containing data from various topical domains, so-called Web of Data.

It is important to distinguish the model itself from its formats. RDF describes only an abstract structure of the data that has to be materialized into a certain format when the data is published on the Web. The first standard serialization format published together with RDF in [7] is RDF/XML. An example of two triples described in RDF/XML format is shown in the listing 1.1. The RDF/XML format suits well the use cases, where little human interaction with the data is expected because its syntax is difficult for a human to read and write compared to other formats. On the other hand, its XML background makes it a perfect format for data processed and generated solely by machines.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <rdf:Description rdf:about="http://example.com/john-johnson">
        <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
        <foaf:name>John Johnson</foaf:name>
    </rdf:Description>
</rdf:RDF>
```

Listing 1.1: Example of RDF data described in RDF/XML format (Source: author)

One of the most used and most human-readable formats is Turtle (Tense RDF Triple Language).[1] It provides various shorthands, enabling to make the representation as brief as possible and thus suitable to be written by hand. The common part of URI strings can be prefixed, so only the decisive end of the URIs has to be stated. The symbol of the semicolon is used to divide pairs of predicate and object belonging to the same subject, so the subject does not have to be repeated. If the described triples share both subject and predicate, a comma can be used to divide the different objects of the triples. Usage of a prefix and the two symbols is shown in an example in the listing 7.5.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix eg: <http://example.com/> .
eg:john-johnson rdf:type foaf:Person ;
                        foaf:name "John Johnson" ;
                        foaf:knows eg:john-jackson, eg:jack-johnson .
```

Listing 1.2: Example of RDF data described in Turtle format (Source: author)

Same as with the relational data model and SQL, RDF also needs a capable language for querying and manipulating the data. For this purposes, SPARQL was designed.[8] Example of a simple SELECT query written is SPARQL is shown in the listing 7.8. Similarly to SQL, the WHERE clause serves to limit the search place from which the result of the query is given. Content of the WHERE clause resembles the Turtle syntax. URIs, that can be bound to variable that occurs in the pattern stated in the WHERE clause, are contained in the output of the query if the variable is enumerated after the SELECT keyword. This

particular query would return all possible bindings for variable `name` ie. all names of persons, for whom the queried data states, that they know a certain John Jackson.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX eg: <http://example.com/>

select ?name where {
    ?person rdf:type foaf:Person ;
            foaf:name ?name ;
            foaf:knows eg:john-jackson .
}
```

Listing 1.3: Example of a simple SPARQL query (Source: author)

1.2 Linked Open Data

The first activities with the goal of starting the publication of Linked Data on a global scale were conducted by the Semantic Web research community as part of the W3C Linking Open Data (LOD) project established in 2007.[6] The aim of the project was to identify datasets published under an open license and to publish them according to the Linked Data principles. All data sets that are published under an open license and are connected to other data sets are referred to as LOD cloud.

The content of the LOD spans across multiple domains. The website lod-cloud.net tracks the current state of published LOD data sets and divides the data sets into these categories, so-called subclouds: Cross-Domain, Geography, Government, Life Sciences, Linquistics, Media, Publications, Social Networking and User-Generated. A data set can fall into more than one category. Cross-Domain, general knowledge data sets play an important role of an intermediary through which unrelated data sets can be connected.

One of those data sets is Wikidata. It is a sister project of Wikipedia, founded and hosted by the Wikimedia Foundation. The data set is managed in an open an collaborative way. Everybody who is interested in expanding the knowledge base can create an account and start contributing. The website of the project provides an intuitive user interface for editing and creating data, so no technical skills beyond common usage of the Internet is needed. The data set currently contains over 93 million items edited by over 26 thousand active contributors. Every item of the dataset is allocated an unique identifier prefixed by the letter **Q**, so-called QID or Q number. The items are described by their statements corresponding to RDF triples. Predicates are in the context of Wikidata called properties and are prefixed by letter **P** similar to items. A sample of the triples contained in Wikidata in Turtle syntax shows listing 1.4.

An different approach is taken by the DBpedia project. Instead of relying on manual contribution of volunteers, DBpedias data comes from an application of NLP extraction algorithms over plain text of Wikipedia's articles.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix wd: <http://www.wikidata.org/entity/> .
@prefix wdt: <http://www.wikidata.org/prop/direct> .
@prefix schema: <http:schema.org/> .

wd:Q111 wdt:P361 wd:Q7879772
       rdf:label "Mars" ;
       schema:description "fourth planet from the Sun";
```

Listing 1.4: Wikidata content sample (Source: author)

2. Data Cubes

While relational databases with highly normalized data models fit well to situations where data is frequently modified, they can be quite cumbersome when being performed complex aggregating queries. Online Analytical Processing (OLAP) system fits better these purposes. In an OLAP system, numerical data is stored in a multidimensional data structure. The structure is comprised of hypothetical cells, which are identified by their assigned set of dimension values from each dimension of the structure. Each cell can contain zero to many numerical values, so-called measurements. The structure is referred to as OLAP Cube or Data Cube. The word *cube* implies exactly three dimensions, but its purpose is only to illustrate the multidimensionality of the structure.

Distinct values of a dimension can be organized into a hierarchy, where a parent value is assigned to summarized measurements throughout its child values. An example of such a hierarchy could be a relationship of a product category and specific products belonging to this category. The depth of a dimension value in its hierarchy then determines the level of granularity the measurement values in a cell assigned to the dimension value are associated with. A cell with all dimension values at the lowest level in their hierarchies or in no hierarchy at all has the finest level of granularity. In a Data Cube consisting of only one cell, meaning each dimension of the cube has only one distinct value, the cell has the coarsest level of granularity.

Several operation can be performed on a Data Cube:

roll-up This operation aggregates data either by reduction of one or more dimensions or by climbing up a concept hierarchy for a dimension.

drill-down This operation transforms data to a more detailed level. It is the opposite of roll-up operation. Either a new dimension is added or the values are projected on a more granular level of a dimension.

slice and dice By slicing a cube only certain subset of the dimension values of one dimension is allowed in the resulting cube. Dicing means restricting dimension values across multiple dimensions.

pivot Pivoting means rotating the cube by its axis in order to change the view of the data.

If the values contained in the cube have an additive character (e.g. sales amount or a number of security incidents), the values can be rolled up or drilled down along any dimension. Not all facts are additive though (e.g. average temperature). The analytical process itself lies in performing the above-mentioned operations in order to find interesting insight into the data. By precomputing the aggregations of all possible subsets of dimensions from the cube on the finest level of granularity, the whole process can be accelerated.

2.1 The Data Cube Vocabulary

The principle of dimensions, measures and attributes are the basic building blocks of the standards and guidelines presented by the SDMX (Statistical Data and Metadata eXchange) initiative, that tries to standardise and modernise the exchange of statistical data. The World Wide Web Consortium's recommendation for representing multi-dimensional data in RDF is the Data Cube vocabulary. This vocabulary underlies the standards and guidelines of the SDMX initiative. It allows to publish the content of the cube together with information about its structure and its metadata. The structure of the vocabulary is shown in the picture 7.1.

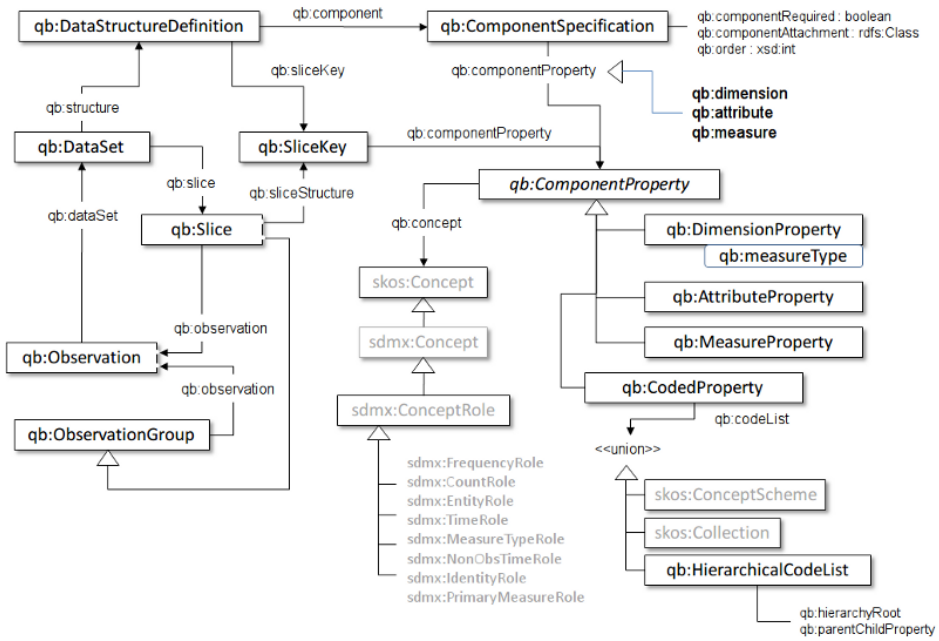

Figure 2.1: The Data Cube Vocabulary structure (Source: [3])

TODO SKOS

```
pk:PensionKindScheme a skos:ConceptScheme ;
    skos:prefLabel "Kinds of pensions"@en .
pkr:PK_VM a skos:Concept, pk:PensionKind ;
    skos:prefLabel "widower's pension"@en ;
    skos:inScheme pk:PensionKindScheme ;
    skos:notation "PK_VM" ;
    skos:altLabel "VM"@cs .
```

Listing 2.1: SKOS

3. Association Rules

4. AMIE Algorithm and Its Derivatives

Finding association rules in knowledge bases can serve several purposes. New facts that are not yet present in the dataset can be derived from the found regularities described by the rules. From such rules opposing facts present in the dataset can be deduced to be wrong. Mined rules can also help to understand the data better.

For mining rules from a graph database such as LOD datasets, Inductive Logic Programming (ILP) can be used. ILP work under the Closed World Assumtion (CWA) meaning it supposed that both negative and positive statements are present in the data. However LOD operates under the Open World Assumption (OWA) ie. if a statement is not present in the data, it does not mean that this statement does not correspond to reality. Rules mined by ILP would not reflect this matter. Moreover ILP are not observed to be efficient over large datasets in the order of millions of statements, making it not a viable way to mine rules over real-world knowledge bases such as YAGO or Wikidata.

4.1 AMIE

An algorithm that is specifically designed to mine rules from data operating under OWA and constisting of binary predicates (just as Linked Data) is AMIE (**A**ssociation Rule **M**ining under **I**ncomplete **E**vidence).[5] AMIE mines rules in the form of Horn rule. Horn rule is an implication with conjunction of atoms on the left side, called body and a single atom on right side call head. We can imagine an atom as an RDF triple, where subject and object can be replaced by variables. Number of atoms in a rule indicates the length of the rule. In this work rules are represented with an infix notation. The AMIE literature use the Datalog notation commonly used in ILP domain. An example of a rule AMIE seeks to discover is shown below.

$$(?a\ worksIn\ ?b) \wedge (?b\ hasHeadquatersIn\ ?c) \Rightarrow (?a\ livesIn\ ?c)$$

This rule states that any person lives in a place his or her company's headquaters. Length of this rule is 3 since it has two body atoms. The rule has 3 variables. When we substitute the variables by constants present in the examined data set, we get an *instantiation* of the rule. If all atoms of the instantiated rule appear in the data set, the head atom of the instantiation is one the *predictions* of the rule. Number of all instantiations of an atom that appear in the data set is called *size* of the atom.

4.1.1 Language Bias

In order to efficiently traverse the search space, AMIE subjects the rules to a particular language bias. Only the rules comforming the conditions stated below can be generated and further refined.

rules have to be connected A rule is connected when every atom in the rule shares every variable with another atom in the rule.

rules have to be closed A rule is closed when every variable appears at least twice in the rule.

rules cannot be reflexive reflexive rule contains at least one atom with identical subject and object variable or constant.

rule can be recursive Any predicate can occur more then once in a rule.

4.1.2 Measures of Significance

Support

For a chosen definition of a support measure for the AMIE algorithm it is crucial for the definition to have the property of monotonicity ie. by adding any new atom to the body of a rule, the support of the rule shall always decrease or remain the same. A naive way to count support of a rule would be to count all instantiations of the rule that appear in KB. Such definition would not comply the property of monotonicity, since an addition of a dangling atom to a rule would introduce a new variable multiplying the number of instantiations and thus the value of the support measure. By counting only all distinct pairs of subjects and objects in the head of all instantiations that appear in KB, the property of monotonicity is preserved:

$$supp(\vec{B} \Rightarrow r(x,y)) := \#(x,y) : \exists z_1 \dots z_n : \vec{B} \wedge r(x,y)$$

Head Coverage

Since the support is an absolute number, so the size of the examined data set has to be taken into account while defining this threshold. Plus If the defined support value is greater than a number of distint triples containing a certain predicate, any rule containing this predicate in the head atom would be disregarded. Head Coverage is the relative expression of support. It is defined as support of a rule over the size of its head atom.

$$hc(\vec{B} \Rightarrow r(x,y)) := \frac{supp(\vec{B} \Rightarrow r(x,y))}{size(r)}$$

4.1.3 Confidence Measures

The above-mentioned measures describe a quantitative significance of the rule in relation to the examined data set. They quantify the true predictions of the rule but do not take into account the false predictions. Confidence is a way to measure the quality of a rule. Generally speaking, confidence is a ratio of true predictions of a rule to the sum of true predictions and the counterexamples. Number of true predictions can easily be expressed by the rule's support. Two different ways to count the counterexamples are discussed below.

CWA and Standard Confidence

Standard confidence considers every fact that is not present in the examined dataset a false fact and thus a counterexample when predicted by a rule. Facts predicted by a rule is either present in the data set or it is not. Therefore the standard confidence is defined as the ratio of the number of true predictions of the rule to the number of all predictions of the rule.

$$conf(\vec{B} \Rightarrow r(x,y)) := \frac{supp(\vec{B} \Rightarrow r(x,y))}{\#(x,y) : \exists z_1 \dots z_n : \vec{B}}$$

This way of generating counterexamples fails to distinguish a false fact from an unknown fact. This conforms the CWA and it is traditionally used for association rule mining over transactional data where this assumption can be applied. For example if the data does not state, that I bought a bottle of milk last Wednesday, then I really did not buy it. AMIE, however, is intended to mine rules from data operating under OWA, so the usage of this measure is inappropriate.

PCA Confidence

For the PCA Confidence, Partial Completeness Assumption (PCA) is used for generating the counterexamples:

If $\langle s\ p\ o\rangle \in KBtrue$ then $\forall_{o'} : \langle s\ p\ o'\rangle)) \in (KBtrue \cup NEWtrue) \Rightarrow \langle s\ p\ o'\rangle \in KBtrue$.

Meaning that if we know any object for given predicate and subject, we know all triples of containing the predicate and subject together. This assumption is certainly true for predicates with high or complete functionality, such as birthdate or capital. A triple predicted by the measured rule is considered an counterexample only when triples with its combination of subject and predicate are present in the data set and none of those has the triple's object.

$$conf_{pca} := \frac{supp(\vec{B} \Rightarrow r(x,y))}{\#(x,y) : \exists z_1 \dots z_n, y' : \vec{B} \wedge r(x,y')}$$

4.1.4 Algorithm

The AMIE algorithm takes as input parameters the RDF graph the rules are to be mined from (KG), the minimum head coverage of the rules ($minHC$), maximal length of the rules ($maxLen$) and the minimal confidence of the rules ($minConf$). For each distinct predicate in the graph a rule with empty body and a head atom with this predicate is created an the algorithm is initialized by filling a queue with those rules.

The algorithm than iteratively dequeues rules from the queue. If the dequeued rules complies the criteria set by the input parameters, it is accepted for output and added to the result rules array. If rule's length is shorter than the maximal stated length, the rule is refined meaning new atoms are added to its body to create new rules. If a new rule reaches the defined minimal head coverage and is not already present in the queue, it is added to the rule queue. The algorithm continues dequeueing the rules until the queue is empty and no more new rules can be created by the refinement.

Algorithm 1 AMIE algorithm

```
1: procedure AMIE(KG, minHC, maxLen, minConf)
2:     queue = [(?a r1 ?b), (?a r2 ?b) ... (?a rm ?b)]
3:     output = ⟨⟩
4:     while ¬queue.isEmpty() do
5:         rule = queue.dequeue()
6:         if AcceptedForOutput(r, out, minConf) then
7:             output.add(rule)
8:         end if
9:         if length(rule) < maxLen then
10:            R(rule) = Refine(rule)
11:        end if
12:        for ri ∈ R(rule) do
13:            if hc(ri ≥ minHC & ri ∉ queue) then
14:                queue.enqueue(ri)
15:            end if
16:        end for
17:    end while
18:    return output
19: end procedure
```

When Choosing larger thresholds on minimal head coverage and minimal confidence and shorter maximum length makes the algorithm stop earlier and generate fewer rules. Choosing smaller thresholds and allowing generation of longer rules results in longer runtime and more found rules. So the setting of these parameters always introduces a trade-off between the runtime and the number of returned rules.

4.1.5 Rule Refinement

By simply enumerating all possible rules and then computing the interest measures for them in order to find the significant ones is not a fiasible approach for large graphs. The exploration of the the search space of the rules has to be done efficiently. Such exploration is performed by the rules refinement. During the rule refinement, the new atoms are added to the rule by

three means (called *operators*). Each operator creates zero to many rules when applied to a rule.

O_D A *dangling* atom is added to the rule. This atom introduces a new variable to the rule. Its second variable is already present in the rule.

O_I An *instantiated* atom extends the rule. Instantiated atom contains a constant and a variable already present in the rule.

O_C A *closing* atom is added. This atom's both variables are already present in the rule.

4.1.6 Querying the Graph

The algorithm uses so-called *count projection queries* to find the predicates and entities with which new atoms are created during the rule refinement by applying the mining operators, such that the minimum head coverage of the new rule is reached. These queries are not efficient when implemented in SPARQL or SQL. So the authors of the AMIE algorithm suggested in-memory database that is tailored to this type of queries.

The data structure constists of six *fact indexes*: each for a permutation of subject (`S`), predicate (`P`) and object (`O`). Let they be denoted as `SPO`, `SOP`, `PSO`, `POS`, `OSP` and `OPS`. Each fact index is a hash table that maps elements of the first column to a hash table that constains elements of the second column as key mapping to a set of third column elements, such that triples in the graph containing the first column element, second column element and the third column element exist. For example the index SPO is a hash table with each subject present in the graph as a key. Every subject key points to a hash table with keys of predicates that exist with this subject in at least one triple in the graph. The predicate key then points to a array of objects for which triples with this subject, predicate and object exist in the graph. That means that each triple in the graph is store six times in this database.

Along with the fact indexes, the *aggregated* indexes are used for the algorithm. There are three (`S`, `P`, `O`) of them for each triple position and they store number of triples in the graph that contain the element of corresponding key. For example the index `P` stores number of triples for each distinct predicate in the graph. The numbers have to be precomputed before initializing the mining. With these indexes, it is easy to check of size of an atom (*size queries*). When computing for example the size of atom `?a knows ?b` the algorithm would turn to the aggregated index P and simply look up the number corresponding to the key `knows`. For an atom with a constant instead of one of the variables, a fact index is used. For computing the size for an atom `?a knows John_Smith` the algorithm would look into the `POS` index and find a hash table corresponding to the predicate `knows`. Then in this hash table it would find the array corresponding to the key of the entity `John_Smith` and count the items in the array. For checking the existence of a triple any fact index can be used. Either way the database allows checking for existence of a triple or computing atom's size in constant time. A drawback of this is that the database is six times memory-demanding than the input graph itself.

The algorithm exercices other types of queries. The size queries are part of the so-called *existence queries* that check for existence of a binding of a conjunction of atoms. The existence queries are in turn used in the so-called *select queries* that look for all instances of a variable in a conjunction of atoms. The select queries are part of the *count queries* that count distinct bindings to variables of an atom in a conjunction of atom. This is useful for computing the confidence of a rule.

For the PCA confidence of a rule the denominator is the bindings count of head atom's variables x and y for which $(x\ r\ y') \wedge \vec{B}$ can be instantiated from the graph. To compute the denominator, the algorithm first fires SELECT DISTINCT x where $(x\ r\ y') \wedge \vec{B}$. Then for each found x it instantiates the conjunction with with x and fires SELECT DISTINCT y where $(x\ r\ y') \wedge \vec{B}$. By adding up the numbers of distinct y from each query the denominator is computed.

Count Projection Queries

The select queries are also necessary for the count projection queries to find the predicates and entities for new atoms for a rule during the rule refinement. The algorithm considers the set minimal head coverage when creating the new atoms to be added. Only the atoms are added for which the new rule still reaches the minimal head coverage. The general structure of a count projection query is as this:

$SELECT\ x, COUNT(H)\ WHERE\ H \wedge \vec{B}$
$SUCH\ THAT\ COUNT(H) \geq k$

where x can represent either predicates or entities of new atoms. $\vec{B}$ is body of the new rule including the added atom. The symbol k stands for an absolute translation of the minimal head coverage for the rule: $minHC \times size(H)$. The authors of the algorithm provide following example for explaining the usage of count projection queries during a rule refinement. The rule $(?x\ marriedTo\ ?z) \Rightarrow (?x\ livesIn\ ?y)$ is about to be refined. When applying a dangling atom operator to this rule, atoms are to be created that contain a new variable $(?w)$. The new variable in the new atoms will be either:

- at the position of subject and the variable $?y$ will be at the position of object: $(?w\ p\ ?y)$
- at the position of subject and the variable $?z$ will be at the position of object: $(?w\ p\ ?z)$
- at the position of object and the variable $?y$ will be at the position of subject: $(?y\ p\ ?w)$
- at the position of object and the variable $?z$ will be at the position of subject: $(?z\ p\ ?w)$

For each of those four *join combinations* a count projection query will be fired to find predicates to be substituted into them:

$SELECT\ p, COUNT(?x\ livesIn\ ?y)$
$WHERE\ (?x\ livesIn\ ?y) \wedge (?x\ marriedTo\ ?z) \wedge (X\ p\ Y)$
$SUCH\ THAT\ COUNT(?x\ livesIn\ ?y) \geq k$

$(X\ p\ Y) \in \{(?w\ p\ ?y),(?w\ p\ ?z),(?y\ p\ ?w),(?z\ p\ ?w)\}$

If the dangling atom operator is applied on an intermediate rule (not all variables are closed) like this one, the dangling atoms are joined on the non-closed variables ($?y$ and $?z$) in this case. If the rule is closed, the dangling atom is joined on all variables in the rule. The closed atom operator would apply join combinations of $(?y\ p\ ?z)$ and $(?z\ p\ ?y)$ for this example. If there were only one open variable, the closed atom operator would apply join combinations with this variable and each other variable in the rule. If there were no open variable, the operator would apply join combinations of all possible pairs of variables in the rule. When applying the instantiated atom operator, first the dangling atom is used to generate new atoms with new variable. For each this new atom a count projection query on instance of the new variable in the atom is fired to find all entities, that can substitute the variable and the minimal head coverage is still reached. Each found entity then forms a new rule with the new atom with the instantiated variable. This is an example of the query for atom $(?x\ citizenOf\ ?w)$:

$SELECT\ w, COUNT(?x\ livesIn\ ?y)\ WHERE$
$(?x\ livesIn\ ?y) \wedge (?x\ marriedTo\ ?z) \wedge (?x\ citizenOf\ w)$
$SUCH\ THAT\ COUNT(?x\ livesIn\ ?y) \geq k$

4.2 AMIE+

In [4] the authors of the AMIE algorithm came up with an improved version reffering to it as AMIE+. The improvements lie in the rule refinement phase and the confidence evaluation. The improvements do not alter the output of the algorithm, they only speed it up so that it is applicable to mining over large knowledge bases.

4.2.1 Rule Refinement

From the refined rule, it is sometimes possible to infere that an application of a operator would not yield better on any new rules. AMIE+ adds these checks before an operator application:

1. When a not-closed rule with a length of $maxLen - 1$ is being refined, the dangling atom operator is not applied, since the new variable cannot be closed before exceeding the $maxLen$ threshold.
2. When a not-closed rule with a length of $maxLen - 1$ and more than two open variables is being refined, the closed atom operator is not applied, because one variable would still be open when the rule exceeds the $maxLen$ constraints so no such rules would be accepted for output.
3. A not-closed rule with a length of $maxLen - 1$ and more than one open variable is not applied an instantiated atom operator on, because the instantiated atom operator can

only close at most one variable.

4. If the refined rule reaches PCA confidence of 1, it is no further refined, since the new rule would not increase in confidence. They can only decrease in support.

Also in some cases a dangling atom cannot reduce support of a rule. It is when the parent rule already contains atoms with the same relation as a dangling atom and these atoms have a variable in common with the dangling atom. The child rules would have the same support as the parent rule, so the support computation for these rules can be skipped.

4.2.2 Speeding Up Confidence Evaluation

The most important improvement is the confidence approximation. Authors state that during experiments with AMIE, 35% of runtime was spent on the confidence compututation. The algorithm has to at first compute the rule's confidence and only after that the rule can be discarded when it does not the reach a $minConf$ threshold. So the algorithm can spend hundreds of milliseconds computing confidence of a rule for nothing. But instead of computing it, the confidence can be estimated, which is much faster (the authors claim 200-fold speed up). The approximation is based on statistics about predicates (functionality and inverse functionality of predicates, size of the joins betweeen predicates etc.) in the input graph, that are precomputed when the input graph is loaded into the in-memory database.

The approximation is designed to overestimate the confidence so that no high confidence rules are pruned. But the approximation does not just simply substitute the exact confidence computation. The approximation is only applied to rules that have intermediate variables (variables that do not appear in the head atom) and there exists a single path to one head variable to the other through the intermediate variables. For example the rule $(?a\ livesIn\ ?b) \wedge (?b\ hasStreet\ ?c) \Rightarrow (?a\ worksIn\ ?c)$ has one intermediate variable $?b$. There is a single path of variables connecting the variables of the rule's head: $?a \rightarrow ?b \rightarrow ?c$. So this rule's confidence would be approximated instead of computed. Rules without intermediate variables are supposed to have smaller number of bindings to their body so the confidence should be quicker to computed. Confidence for ules with more than single path connecting the head variables (such as $(?a\ livesIn\ ?b) \wedge (?b\ hasStreet\ ?c) \wedge (?a\ bornIn\ ?d) \wedge (?d\ hasStreet\ ?c) \Rightarrow (?a\ worksIn\ ?c)$ that has two paths: $?a \rightarrow ?b \rightarrow ?c$ and $?a \rightarrow ?d \rightarrow ?c$) is also thought to be easy to compute because more variable paths tend to restrict the number of head variables bindings. If the estimated value reaches the $minConf$ threshold, the exact confidence value is computed as well.

4.3 RDFRules

Although the AMIE+ algorithm made it possible to mine rules from large real-world knowledge bases, it lacks features that would allow it to encompass the whole mining process. It does not deal with pre-processing the input data and post-processing of the generated rules.

Those tasks are assumed to be subject to other tools. Modern algoritmic frameworks for mining association rules from tabular data, such as *arules* R package or Spark MLlib,however enable to deal with the whole mining process with just one tool.

TODO prepsat to

4.3.1 Processing of Numerical Attributes

AMIE+ treats numerical values (in RDF those can only appear at the position of object) just as any other distrete value. Due to the downward closure property of the algorithm (and of the association rule mining algorithms in general), where not only the whole rule but also each subset of the rule's atom have to meet the minimum support threshold, and due to the fact that the numerical attributes usually contain many distinct values, the rules concerning numerical values tend to be excluded from the generated rules.

This problem can be solved by the values' discretization meaning that the continuous values are converted to a finite set of intervals. When working with transactional data such sets of intervals are created within each examined table's column which contains numerical values. For example the *arules*[1] R package implementing the Apriori algorithm enables discretizing[2] the values either equidistantly (the intervals have a given fixed length), equifrequently (a given number of intervals are evenly represented in the continuous data points), by clustering the values to a given number of intervals or by stating the intervals manually.

In the context of RDF the discretization entails grouping numerical values in the range of a certain predicate. This can also be done equifrequently, equidistantly etc. However, when the created intervals are too narrow, the rules containing the discretized predicates may not satisfy the minimum support or the head coverage threshold defined for the mining task. When creating too broad intervals the found rules may be unnecessarily general. That means that the discretization phase cannot easily be decoupled from the mining phase itself.

In [10] the authors propose a discretization heuristic that takes into account the minimum head coverage and minimum head size thresholds to perform a specific discretization for a particular task in the pre-processing phase. For each predicate in the data set that has a numerical range a set of overlapping intervals is generated. For each interval in this set and for each predicate's triple with the number in the range of the interval a copy of the triple whose object is substituted and whose predicate is altered by a suffix of the interval set (not to modify the range of the original predicate) by the interval is added to the data set.

[1] https://www.rdocumentation.org/packages/arules/versions/1.6-8
[2] https://www.rdocumentation.org/packages/arules/versions/1.6-8/topics/discretize

4.3.2 Multiple Graphs

Restricting the mining task on to a single graph prevents finding relations accross multiple sources. Even though the resoures representing the identical object or a concept in the real world have a different identifier in different graphs, the identity can be easily infered from the `owl:sameAs` predicate joining two identical resources.

$$(?a\ \langle wasBornIn \rangle\ ?b\ \langle YAGO \rangle) \Rightarrow (?a\ dbo:deatchPlace\ ?b\ \langle DBpedia \rangle)$$

AMIE+ does not recognize the `owl:sameAs` statements and assumes that the identical resources have the same IRI. To mine from accross multiple graphs the graph would either have to be merged together and their IRIs would have to be unified or the `owl:sameAs` relations would have to be explicitely stated in the rules.

RDFRules natively supports *quads* i.e triples enriched with IRI of their corresponding named graph. On top of the AMIE's six indeces four new ones are added (**PG**, **PSG**, **POG**, **PSOG**) that allow to check the affiliation of triples to a graph. Not only the individual triples but also the atoms in a rule can be treated as quads. The atoms can be restricted via the rule patterns (see 4.3.3) to be based only on triples from a certain graph. The RDFRules algorithm recognizes the `owl:sameAs` triples and the resources joined this way are treated as if they had identical identifiers.

4.3.3 Improvements to Expressivenes of Rule Patterns

The association rules mining algorithms tend to create rules describing obvious and uninteresting patterns when the search space of the rules is not properly restricted. AMIE allows the user to provide a list of predicates that should be included or excluded in the rules' body and head and also to prohibit or enforce contants, so that the shape of the rules can be controlled to some extent. RDFRules offers much more profound solution in the form of a rule pattern grammar.

The user defines rule patterns which are used to prune the rules during the rule refinement. Each rule has to match at least one of the rule patterns defined for the mining task. A rule pattern consists of atom patterns corresponding to atoms of a matching rule. The rule pattern can have zero or any number of atom pattern in the body and exactly one atom pattern in its head. A rule pattern with atom patterns in the body and no atom pattern in the head is useless since the rule patterns are applied from right to left just as the rule refinement operators. An atom pattern consists of four atom item patterns that correspond to subject, predicate, object and graph of the matching rule atom. Those atom item patterns are available:

? pattern for any symbol at the position

$?_v$ pattern for any variable
$?_c$ pattern for any constant
$?a$ pattern for a concrete variable written as a single alphabetic character after the symbol ?
[ ] collection of items where at least one has to be matched at the position, that can be also negated by the symbol ¬ stating that none of the items must match at the position

Shown below is an example of a valid rule pattern and its matching rule:

$$(?a\ rdf:type\ ?_c) \wedge (?a\ ?\ ?_v) \Rightarrow (?a\ ?\ ?)$$

$$(?a\ rdf:type\ dbo:Scientist) \wedge (?a\ dbo:academicDiscipline\ ?b) \Rightarrow (?a\ dbo:knownFor\ ?b)$$

4.3.4 Top-k Approach

RDFRules algorithm offers an alternative to manually defining an interest measure (e.g. minimum support or minimum confidence) for the mining task. Instead, in the so-called *top-k approach*, the user defines a maximum number of rules with the highest values of a chosen measure that should be returned by the algorithm. During the mining process, the rules are stored and sorted in a queue with a fixed length of the defined number. The head of the queue contains a rule with the lowest value of the chosen measure. This value acts as a temporary minimum threshold. Once a rule with a higher value of the measure is found, the head rule is removed from the queue, the new rule is added to the queue and the queue is sorted again so that the head of the queue still contains the rule with the lowest value. At the end of the mining process only the rules in the queue are returned.

4.3.5 Support for the Lift Measure

The lift is a measure that describes how the probability of the consequent (head) occurrence is increased given that the antecedent (body) of the rule is valid compared to its probability of occurence under a random choice in the complete dataset. RDFRules adds support for this measure. It is computed as a ratio of the rule's confidence and its *head confidence*.

$$lift(\vec{B} \Rightarrow H) = \frac{conf(\vec{B} \Rightarrow H)}{hconf(H)}$$

The formula of the head confidence depends on the type of the head atom. If the head atom contains two variables ($H = (?a\ p\ ?b)$), the head confidence is computed as the ratio between the number of distinct subjects bound with the predicate in the head atom and the number of distinct subjects in the whole data set.

$$hconf(\vec{B} \Rightarrow H) = \frac{\#s : \exists\langle s,p,o\rangle \prec (?a\ p\ ?b)}{\#s : \exists\langle s,p,o\rangle \prec (?a\ ?r\ ?b)}$$

If the head atom contains one variable and a constant at the position of object ($H = (?a\ p\ C)$), then the head confidence is computed as the ratio between the number of distinct subjects bound with the predicate in the head atom and with the constant in the head atom and the number of distinct subjects in the whole data set.

$$hconf(\vec{B} \Rightarrow H) = \frac{\#s : \exists\langle s,p,o\rangle \prec (?a\ p\ C)}{\#s : \exists\langle s,p,o\rangle \prec (?a\ p\ ?b)}$$

If the head atom contains a constant at the position of subject ($H = (C\ p\ ?a)$), the formula goes as the ratio of between the number of distinct objects bound with the predicate in the head atom and with the constant in the head atom and the number of distinct subjects in the whole data set.

$$hconf(\vec{B} \Rightarrow H) = \frac{\#s : \exists\langle s,p,o\rangle \prec (C\ p\ ?a)}{\#s : \exists\langle s,p,o\rangle \prec (?a\ p\ ?b)}$$

4.3.6 Rule Clustering and Pruning

When an association rule mining algorithm generates a high number of overlapping rules, a clustering algorithm can be used to group the rules, so that the rules can be presented in a more compact and organized way e.g. as a result of an exploratory analysis. In [10] propose an approach to determine the similarity of two rules based on their content and their interest measures. When comparing the content similarity of rules U and V, where $|U| \geq |V|$ the similarity is computed as the maximum average of atom similarities between atoms of the rule V and a k-permutation of the atoms from the rule U where $k = |V|$.

$$sim_c(U,V) = \frac{1}{|V|} \max_{T \in P(U,|V|)} \sum_{i=1}^{|T|} sim_a(t_i,v_i)$$

The similarity of two atoms is based on the similarities between their subjects, predicates and objects.

$$sim_a(A_1, A_2) = \frac{1}{3}[\, sim(\langle s_1, p_1\rangle, \langle s_2, p_2\rangle) + sim(\langle o_1, p_1\rangle, \langle o_2, p_2\rangle) + sim(p_1, p_2)]$$

The similarity function comparing two predicates returns the value 1 if the predicates are identical and 0 otherwise. The similarity function comparing two subjects (and objects analogously) returns the value 1 either if the subjects are identical and they are not variables or if they are both variables and the predicates of the two rules are identical, it returns the value

0.5 if the subjects are not identical, however the predicates are identical and exactly of the the subjects is a variable, and it returns 0 otherwise.

To avoid a situation where a single triple in the data set is covered by multiple rules returned by the algorithm, the authors of RDFRules suggest adapting the *data coverage pruning* in the post-processing phase. The rules are ranked based on the following criteria:

1. $conf(A) > conf(B)$
2. $conf(A) = conf(B) and hc(A) > hc(B)$
3. rule A has a shorter body than the rule B

Then for each rule starting with the highest ranked to the lowest ranked rule it is checked whether the rule covers at least one triple (i.e. any atom in the rule can be instantiated by the triple) that the previous rules did not and only those rule which satisfy the conditions are kept. That way the new set of rules covers exactly the same set of triples in the data set.

5. RDFRules Reference Implementation

An open-source implementation of the improved AMIE+ algorithm proposed in [10] is available under the name RDFRules at https://github.com/propi/rdfrules.

The core of the framework is written in Scala. Beside the Scala API[1], the framework also provides a Java API[2] serving as the facade into the Scala core, though it is already[3] pronounced deprecated. Both APIs are published[4] in the JitPack package repository, so they can be easily added to a Scala or Java project as a dependency. The framework also has a REST API wrapping the Scala core that can either be used as is and be accessed through an HTTP client or through a GUI via a web broswer.

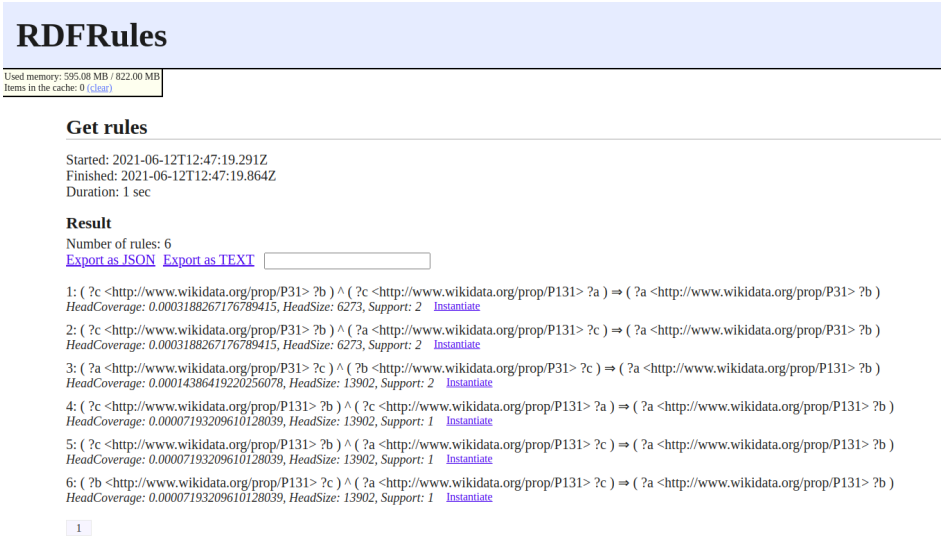

Figure 5.1: RDFRules web browser interface

The framework's core revolves around four main data structures: *RDFGraph* and *RDFDataset* corresponding to the input data, *Index* wrapping the set of indeces the input data is transformed into and *RuleSet* containing rules generated by the algorithm. These structures are transformed into each other in the stated order during the mining process by applying various operations on them. Inspired by the Apache Spark RDD[5] the operations are categorized into *transformations* and *actions*. All transformations are *lazy* operations meaning they are not performed right after the API call but only when an action operation is called which is dependent on them.

Each transformation method called on an instance of the data structures returns either a

[1] https://github.com/propi/rdfrules/tree/master/core
[2] https://github.com/propi/rdfrules/tree/master/java
[3] The current version of the framework as of writing this is *1.5.0* published on November 28th 2020.
[4] https://jitpack.io/#propi/rdfrules
[5] https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds

modified version of the same instance or the instance transform into a further data structure but the instance which the method was called on does not change i.e. the objects of the framework are immutable and the whole workflow with the core API lies in chaining those methods. Multiple calls of an action operation with different input parameters would result in repetitive performance of the defined transformations so each data structure has a *cache* method that enables preserving the result of the transformations in memory or on the disk so that they have to be performed only once.

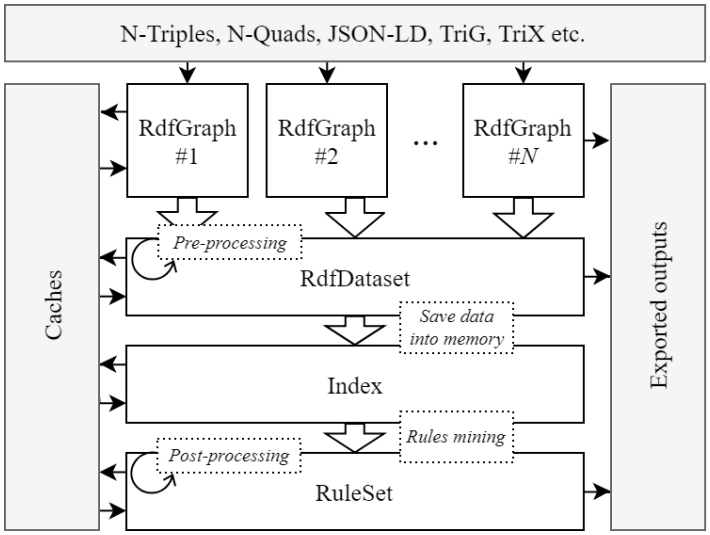

Figure 5.2: Relations between the RDFRules data structures (Source: [9])

An RDFGraph instance corresponds to a set of triples. It is created either from stream of triples or from a file containing a supported RDF serialization (N-Triples, N-Quads, JSON-LD, TriG, Turtle etc.) and optionally can be assigned a graph IRI. RDFDataset can either be created from one or multiple instances of RDFGraph or directly from a file if the input format contains a set of quads so the RDFDataset corresponds to a set of quads. Both RDFGraph and RDFDataset allow filtering, slicing or modifying the data at the level of individual triples/quads. It is possible to merge two instances of RDFGraph, two instances of RDFDataset and to merge an instance of RDFDataset and RDFGraph which results in a new instance of RDFDataset.

Both have a *discretize* method that facades into the EasyMiner-Discretization[6] library. It takes a parameter specifying the kind of discretization (*discretization task*) and a parameter in a form of a function that specififfies which triples are to be processed. It allows to create a defined number of equidistant and equifrequent intervals or to create equifrequent intervals with a defined minimum frequency (*equisize* discretization task).

The RDFDataset instance is transformed to an instance of Index on which the rules can be mined with. During this transformation, each resource and literal is assigned a unique number which reprents it in the created indeces. The mining itself, which is triggered by a *mine* method is controlled by defined *pruning thresholds* (pruning during the rule refinement), rule

[6]https://github.com/KIZI/EasyMiner-Discretization

patterns and *constraints*. The available pruning thresholds are minimal head size, minimal head coverage, minimal support, maximum rule length and *timeout* (a time period after which the mining is stopped and so far found rules are returned) in minutes. It also implements the TopK pruning threshold mentioned in [10] although only for the head coverage. It does not allow pruning the rules based on minimum confidence as the AMIE+ implementation of Galárraga et al. does. The constraints can specify general characteristics of the rules that can be returned e.g. disallowing duplicit predicates or any constant in the rules.

The rules are returned by the *mine* method as an instance of RuleSet. Each rule in the set basic measures such as support, head size etc. and other *computationaly expensive* measures such as confidence (PCA or a standard confidence) can be calculated optionally. The rules in the set can be filtered and sorted by those measures. Pruning and clustering can be performed on the set. The algorithm used for clustering is DBScan which takes parameters of minimum size of a cluster, minimum similarity of rules in the same cluster and the weights on the similarity features (whether the clustering should be more based on the rule contents of measures). The rules can be exported into a text file as a human-readable text or in JSON format. Since the mining algorithm and the RuleSet structure do not work with IRIs and literals directly but with their IDs assigned during the creation of indeces, when the rule set is cached into a file, the appropriate instance of Index is needed to restore the rule set, so that the IDs from which the rules consist can be mapped to their IRIs and literals (the rules can be *resolved*).

```
Dataset("yago.tsv")
    .filter(!_.triple.predicate.hasSameUriAs("participatedIn"))
    .discretize(DiscretizationTask.Equifrequency(3))
    (_.triple.predicate.hasSameUriAs("hasNumberOfPeople"))
    .mine(Amie()
    .addConstraint(RuleConstraint.WithInstances(true))
    .addPattern(AtomPattern(predicate = Uri("hasNumberOfPeople")) =>: None)
    .addPattern(AtomPattern(predicate = Uri("hasNumberOfPeople"))))
    .computePcaConfidence(0.5)
    .sorted
    .export("rules.json")
```

Listing 5.1: An example of a rule mining workflow with RDFRules Scala API (Source: [9])

6. Leveraging a Combination of OLAP Cubes and Knowledge Graphs

6.1 Mining from RDF representation of Data Cube

6.2 Appending RDF Data to the Data Cubes

7. Experiment

This section describes an experiment of mining associtation rules from RDF data compiled of statistical data structured by the Data Cube Vocabulary and facts pulled from the Wikidata data set that was performed as an practical part of this work. The statistical data come from two sources. The first one is the Czech Social Security Administration and the second one is the Czech Statistical Office. Analysis was performed using the Scala API of the reference implementation of the RDFRules algorithm. The following sectins describe how the available data had to be preprocced to give reasonable results in combination with KG data. The preprocessing was perfomed partly by the implementation's API itself, partly by performing SPARQL queries over the data. The described method can be taken inspiration from when performing similar analysis ie. association rule mining task over the multidimensional data merged with loosely structured graph data.

7.1 Czech Social Security Administration's Data Cubes

Czech Social Security Administration (CSSA) is a czech public administration organtisation responsible for collecting social security premiums and contributions to the state employment policy. Since 2015 the organization publishes its statistical yearbook datasets and other (vocabularies, code lists and datasets containing data concerning the internal operation of the organization) in the form LOD and became of the first czech public institutions to do so. The yearbook statistical data sets are modelled using Data Cube Vocabulary. Their dimension values are represented by the SKOS vocabulary. The organization has published 73 datasets so far. All these datasets are downloadable as dumps[1] or accesible througth a SPARQL endpoint[2]. The CSSA's URI are dereferenceable.

The largest of the data cubes published is `cssa-d:duchodci-v-cr-krajich-okresech`[3]. From now on it will be denoted as CSSA1. It contains 368 118 observations structured spread over four dimensions: reference area[4], reference period[5], sex[6] and pension kind[7]. Observations are assigned three measures: the average amount of pension[8], the average age[9] and the number of persons[10]. Each observation is assigned only one measure.

In this particular data set there are 102 distinct values of the dimension of reference area:

[1] http://data.cssz.cz/web/otevrena-data/katalog-otevrenych-dat
[2] http://data.cssz.cz/web/otevrena-data/sparql-query-editor/
[3] https://data.cssz.cz/resource/dataset/duchodci-v-cr-krajich-okresech
[4] https://data.cssz.cz/ontology/dimension/refArea
[5] https://data.cssz.cz/ontology/dimension/refPeriod
[6] https://data.cssz.cz/ontology/dimension/pohlavi
[7] https://data.cssz.cz/ontology/dimension/druh-duchodu
[8] https://data.cssz.cz/ontology/measure/prumerna-vyse-duchodu-v-kc
[9] https://data.cssz.cz/ontology/measure/prumerny-vek
[10] https://data.cssz.cz/ontology/measure/pocet-duchodcu

```
@prefix qb: <http://purl.org/linked-data/cube#> .
@prefix cssa-om: <https://data.cssz.cz/ontology/measure/> .
@prefix cssa-d: <https://data.cssz.cz/resource/dataset/> .
@prefix cssa-od: <https://data.cssz.cz/ontology/dimension/> .

<https://data.cssz.cz/resource/observation/duchodci-v-cr-krajich-okresech/2017-12-31/
     prumerna-vyse-duchodu-v-kc/pk_srnvm/vc.35/m>
    a qb:Observation ;
    qb:dataSet cssa-d:duchodci-v-cr-krajich-okresech .
    qb:measureType cssa-om:prumerna-vyse-duchodu-v-kc ;
    cssa-od:druh-duchodu <https://data.cssz.cz/resource/pension-kind/PK_SRNVM_2010> ;
    cssa-od:pohlavi <https://data.cssz.cz/ontology/sdmx/code/sex-M> ;
    cssa-od:refArea <https://data.cssz.cz/resource/ruian/vusc/35>;
    cssa-od:refPeriod <https://data.cssz.cz/resource/reference.data.gov.uk/id/gregorian-day/2017-12-31>;
    cssa-om:prumerna-vyse-duchodu-v-kc 6622.0;
```

Listing 7.1: Example of an observation from CSSA1

14 regions (NUTS 3 administrative units, czech translation in singular nominative is *kraj*) including Prague, 77 districts (*okres*) also including Prague, 10 Prague districts (*správní obvod*) and a value representing the state in total. Each entity representing a reference area is assigned an unique numerical identifier which corresponds to this area's identifier in the official Registry of Territorial Identification, Addresses and Real Estate (RTIAR) runned by the State Administration of Land Surveying and Cadastre (SALSC). RTIAR codes are reference codes by law, so it is obligatory for CSSA to use them and have them correct.

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

<https://data.cssz.cz/resource/ruian/vusc/35>
    a <https://data.cssz.cz/ontology/ruian/Vusc> , skos:Concept ;
    <http://www.w3.org/2002/07/owl#sameAs> <https://linked.cuzk.cz/resource/ruian/vusc/35> ;
    skos:inScheme <https://data.cssz.cz/resource/ruian/ConceptScheme> ;
    skos:notation "VC.35" ;
    skos:prefLabel "Jihočeský kraj" .
```

Listing 7.2: Dereferenced proxy entity of the South Bohemian Region

There are official URIs of this registry but CSSA datasets do not used them directly. The entities for the dimension of reference area and other dimensions in the dataset CSSA1 and all other data sets of CSSA with the dimension of reference area work as so-called *proxy entities*. This means that instead of using the original code list item URIs directly as objects in the RDF triples, it uses their equivalents defined in the internal code lists. These equivalents are connected to the original URI by the `owl:sameAs` statement. These proxies can then contain data specific to CSSA e.g. labels. Another advantage of this is, that these URIs are dereferenceable to the CSSA domain and their versioning is under the control of CSSA and they can be easily redirect to a different equivalent code list. Previously the proxy entities of the reference area were directed to the unofficial transformation of the Opendata.cz initiative[11].

[11] https://linked.opendata.cz/dataset/cz-ruian

Another dimension whose values work as proxy entities is the dimension of reference period. This dimension divides the observations into one year intervals. This applies to all other CSSA data cubes containg the reference period dimension. The data set vary in the overall covered period. The last covered year of all data sets is the year 2019. The entities link to the `data.gov.uk` Time Intervals[12] OWL ontology. Usage of this entities is, however, not unified. In some data sets intervals are assigned an entity representing a year, in other they are assigned an entity representing the last day of the corresponding year. All of them are, however, representing a period of a whole year.

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

<https://data.cssz.cz/resource/reference.data.gov.uk/id/gregorian-year/2017>
    a skos:Concept ;
    <http://www.w3.org/2002/07/owl#sameAs> <http://reference.data.gov.uk/id/gregorian-year/2017> ;
    skos:inScheme <https://data.cssz.cz/ontology/years/YearsScheme> ;
    skos:notation "2017" ;
    skos:prefLabel "2017" .
```

Listing 7.3: Dereferenced proxy entity of the year 2017

The CSSA1 dataset uses two distinct schemes for the pension kinds, because in 2010, the official categorization of pensions was changed in the czech legislation. Only the observations assigned to year 2008 are divided according to the old pension scheme. All other year's observations correnspond to the new pension scheme. So one can not simply multiply the numbers of distinct values for each dimension and the number of measures to get the total number of observations for this particular cube. The URIs of both pension kind schemes are suffixed either by `_2008` (31 of them) for the old scheme or `_2010` (37 of them) for the new scheme. Not all entities correspond to a particular kind of pension. Some of them represent an aggregation over related pension kinds or simply an aggregation over all of the pension kinds.

```
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .

<https://data.cssz.cz/resource/pension-kind/PK_SRNVM_2010>
    a <https://data.cssz.cz/ontology/pension-kinds/PensionKind> , skos:Concept ;
    skos:altLabel "SRNVM"@cs ;
    skos:exactMatch <https://data.cssz.cz/resource/pension-kind/PK_SRNVM> ;
    skos:inScheme <https://data.cssz.cz/ontology/pension-kinds/PensionKindScheme_2010> ;
    skos:notation "PK_SRNVM" ;
    skos:prefLabel "Starobní důchod SRN vyplácený v souběhu s vdoveckým důchodem"@cs .
```

Listing 7.4: Dereferenced pension kind

The dimension of sex consists of three distinct values: dimension of male pension, dimension of female pensions and its total. The values are proxy entities linking to the SDMX representations of sexes[13]

[12] http://old.datahub.io/dataset/data-gov-uk-time-intervals
[13] http://purl.org/linked-data/sdmx/2009/code

7.2 Czech Statistical Office's Data Cubes

The Czech Statistical Office (CZSO) is the main public organization responsible for collecting and analyzing statistical data in the Czech Republic. This organization is for example responsible for the state's census. Data about demography, economics, education, health care etc. are made availabe on the organization's website[14] in a form of interactive spreadsheet builder. Thanks to the Opendata.cz initiative this data sets are made available as LOD modelled by the Daca Cube Vocabulary in the initiative's catalogue. The data is also hosted as a SPARQL endpoint[15]. 8 of these data sets have a dimension of reference period. Each data set's dump file can be downloaded from the catalouge[16].

```
@prefix qb: <http://purl.org/linked-data/cube#> .
@prefix czso: <http://data.czso.cz/ontology/> .

<http://data.czso.cz/resource/observation/job-applicants-and-unemployment-rate/CZ0513/2009-12-31/T> a
     qb:Observation ;
    czso:refArea <http://ruian.linked.opendata.cz/resource/okresy/3505> ;
    czso:refPeriod <http://reference.data.gov.uk/id/gregorian-day/2009-12-31> ;
    czso:sex sdmx-code:sex-T ;
    czso:neumisteniUchazeciOZamestnani 9692.0 ;
    czso:dosazitelniNeumisteniUchazeciOZamestnani 9528.0 ;
    czso:podilNezamestnanych 7.92 ;
    czso:pocetVolnychMist 569.0 ;
    qb:dataSet <http://data.czso.cz/resource/dataset/job-applicants-and-unemployment-rate> .
```

Listing 7.5: Example of an observation from the CZSO data sets

Observations in CZSO data cubes are assigned multiple measures. Their URIs are not dereferenceable. Their dimension value URIs do not work as proxy entities. The dimension of reference area uses entities of an above-mentioned initiative's unofficial RTIAR transformation[17] with its own SPARQL endpoint[18]. The measured values relate to regions and district. They do not contain observations related to the whole state. The proxy entities of the reference area dimension values for the CSSA data sets previously linked to this code list.

For time intervals representation the CZSO data cubes also use the the `data.gov.uk` Time Intervals OWL ontology. They only do so directly unlike the CSSA data cubes. The data cubes vary their time span. The earliest recorded values are for the year 2005. The latest values are for the year 2013. There are two data sets that contain values for both the earliest and latest year mentioned meaning they cover a period of 9 years: **czso-deaths-by-selected-causes-of-death**[19] and **czso-job-applicants-and-unemployment-rate**[20]

[14] https://vdb.czso.cz/vdbvo2/faces/en/index.jsf?page=uziv-dotaz

[15] http://linked.opendata.cz/sparql

[16] The download link URLs are, however, broken and return HTTP status code 404. To get the dump file, word *dumps* has to be substituted with word *soubor* (czech word for *file*). For example, the dump file of the data set **czso-job-applicants** is available at https://linked.opendata.cz/soubor/czso-job-applicants.trig

[17] https://linked.opendata.cz/dataset/cz-ruian

[18] https://ruian.linked.opendata.cz/sparql

[19] https://linked.opendata.cz/dataset/czso-deaths-by-selected-causes-of-death

[20] https://linked.opendata.cz/dataset/czso-job-applicants-and-unemployment-rate

Just as with the CSSA data cubes, some of the CZSO data cubes contain the dimension of sex consisting of three distinct values: dimension of male pension, dimension of female pensions and its total. The values used are the SDMX representations of sexes themselves.

7.3 Wikidata

Wikidata data set contains data about political representation of countries, their administrative areas and municipalities. For the purposes of this experiment, such data concerning the Czech Republic was extracted from the data set. In the Czech Republic, regions and municipalities[21] are being assigned a government that emerges from elections. In Wikidata data set, there exist records of who was or still is head of this local government, including the head of the state government. Records of these *head of government* roles are given a time period of validity of this role by stating the date of this role's start and optionally the end of this role when it is not a current area government head anymore. For the persons who hold or held the office, the affiliation to a political party is stated in the data also with the start and end date of this affiliation. The entities of the political parties are assigned their political alignment (left, center, far-right etc.). In the sample of Wikidata data set's content below it is stated that since 2008 till 2016 the head of the government of the South Moravian Region was Michal Hašek who since 1998 is a member of the Czech Social Democratic Party, which has the centre-left political alignment.

```
@prefix wd: <http://www.wikidata.org/entity/> .
@prefix p: <http://www.wikidata.org/prop/> .

wd:Q192697 rdfs:label "South Moravian Region"@en ;
    p:P6 [ p:P6 wd:Q6835752 ; p:P580 "2008-11-21T00:00:00Z" ; p:P582 "2016-11-16T00:00:00Z" ] .

wd:Q6835752 rdfs:label "Michal Hašek"@en ;
    p:P102 [ p:P102 wd:Q341148 ; p:P580 "1998-01-01T00:00:00Z" ] .

wd:Q341148 rdfs:label "Czech Social Democratic Party"@en ;
    p:P1387 [ p:P1387 wd:Q737014 ] .

wd:Q737014 rdfs:label "centre-left"@en .
```

Listing 7.6: Description of XXX (Source: author)

When adequately preprocced, this data can be utilized to find relations of statistical data described in the data cubes of CSSA and CZSO and the political cycle in the country. For example a rule can be found that states, that if in any year, the head of the Czech Republic's government was a member of a left-leaning policital party, the pension expenses for one-off allowance to pensions were above average. A query that extracts this data from the Wikidata's SPARQL endpoint is listed in A.2. This data, however, cannot be used for mining such rules yet. Measures in the data cubes are recorded on year to year bases. For the governmental roles and political affiliations it is only know the start date and end date.

[21] Not districts though.

It is necessary to transform these triples into set of triples stating that a governmental role or the policital affiliation *applies* for a certain year. The edge years are a bit tricky because the role or the membership was not valid for the whole year it started or ended. To facilitate the query and to generate more triples I decided to generate the triples for the edge years as well. The SPARQL query that constructs the *appliesToRefPeriod* triples from the extracted data is listed in A.3. The triples stating the start dates and end dates are no longer needed and do not have to be loaded into the RDFRules mining task.

The triples with predicates of `p:P582` and `p:P580` are no longer needed and are filtered out during the preproccesing. That will leave the data with 24 410 distinct triples.

7.4 YAGO Triples

Triples concerning the entitities of reference areas in the examined data cubes can also be found in the YAGO[22] dataset, which extracts facts about real world entities from various sources (Wikipedia, GeoNames, WordNet etc.). For a change unlike with the Wikidata dataset, for this experiment the needed triples are not extracted according to a rigidly structured CONSTRUCT queries but with loose DESCRIBE queries returning all triples concerning the seeked entities. The used queries[23] extract the triples containing the reference areas themselves, the entities appearing in the triples together with the reference area entities (*hop 1*) and the entities appearing in the triples of those entities (*hop 2*). YAGO's public SPARQL endpoint's web interface[24] has trouble handling the volume of the triples returned in the latter mentioned queries so one is better off quering the endpoint directly through a HTTP client.

Both districts and regions of the Czech Republic have their class[25][26] of which they are subclass in the YAGO dataset so the needed triples are easy to query. Total of 2 992 361 distinct triples was extracted, containing data about persons connected to the areas, geographical information about the areas (for example which region contains which district) etc.

```
@prefix schema: <http://schema.org/> .
@prefix yago: <http://yago-knowledge.org/resource/> .

yago:Prague a yago:Capital_city .
yago:Josef_Jiří_Stankovský_Q12026167 schema:deathPlace yago:Prague .
yago:What_the_Old_Man_Does_is_Always_Right_Q13564487 schema:translator yago:Josef_Jiří_Stankovsk
    ý_Q12026167
```

Listing 7.7: Example of the Extracted YAGO Triples (Source: author)

[22] https://yago-knowledge.org/
[23] https://github.com/nvkp/diploma-thesis-code/tree/master/data/yago
[24] https://yago-knowledge.org/sparql
[25] http://yago-knowledge.org/resource/Districts_of_the_Czech_Republic
[26] http://yago-knowledge.org/resource/Regions_of_the_Czech_Republic

Around 62% of the extracted triples are not relevant for the performed tasks. Their objects are literals (`rdfs:label`, `rdfs:comment` and `schema:alternateName`) or image URLs (`schema:image`). These triples are filtered out before the index data structure is constructed.

7.5 Filtering the Observations

The values for the city of Prague are duplicated. The city is assigned an entity both as a region and as a district. The administrative area of the Czech Republic's capital is given a special status by the Act No. 131/2000 Coll., on the Capital City of Prague and does not in fact fall into neighter of those categories. Nonetheless, Prague is assigned an identifier both as a district (3 100) and as a region (19) in the RTIAR registry. When it comes to total population of the area (1 324 227 as of 2020), it is comparable to other czech regions. The least populated is the Karlovy Vary Region with around 300 000 inhabitants and the most populated: the Central Bohemian Region has around 1 300 000 inhabitants. Its surface area is on the other hand comparable with the districts. As the statistics about pensioners are certainly more correlated with the population rather than the surface area, the observation allocated to the dimension value of Prage as being district would be filtered out to maintain the commensurability along values measured for districts (see 7.6).

I also chose to filter out all observations regarding year 2008. For 2008 the pension kinds are structured according to a different scheme than any other year and it is hard to assume compatibility for the URIs that only differ in the year's suffix. 2008 scheme contains penkind kinds that 2010 does not end vice versa. It could be possible to just cut the cube so that the year's 2008 become one cube and the other years the other one, but a cube concerning only one reference period has not got much value. Both filters can be performed in a single SPARQL query. In the CSSA1 data set, discarting the Prague as a District entity removes 3 609 observations. The year 2008 contained 28 458 observations. 336 330 observations are contained in the query's result making up 91,4% of the unfiltered data set.

7.6 Slicing the Cubes

It is in part aimed to mine rules, in which the head atom's predicate is one of the cube's measures. In order to ensure, that such rules can achieve a reasonable support, the numerical values at the position of object in the measure triples have to be discretized and replaced by intervals. Irrespective to a chosen discretization approach, it is inadmissible to discretize values belonging to different disproportionate contexts. For example, we cannot create intervals for the number of pensioners from values measured for both regions and districts together. A district is a lower administrative unit. It belongs to a lower level in the concept hierarchy and it is assumed that its numbers of pensioners are of a different order of magnitude than those for regions or for the whole state. Same applies for values of dimensions sex and pension kind

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX qb: <http://purl.org/linked-data/cube#>
PREFIX cssa-d: <https://data.cssz.cz/resource/dataset/>
PREFIX cssa-od: <https://data.cssz.cz/ontology/dimension/>
PREFIX cssa-rd: <https://data.cssz.cz/resource/ruian/okresy/>
PREFIX cssa-op: <https://data.cssz.cz/ontology/pension-kinds/>

CONSTRUCT {
    ?observation ?p ?o
}
WHERE {
    GRAPH cssa-d:duchodci-v-cr-krajich-okresech {
        ?observation qb:dataSet cssa-d:duchodci-v-cr-krajich-okresech ;
            cssa-od:druh-duchodu ?druh ;
            ?p ?o .
        NOT EXISTS {
            ?observation cssa-od:refArea cssa-rd:3100 .
        }
     }
     GRAPH cssa-d:pomocne-ciselniky {
        ?druh skos:inScheme cssa-op:PensionKindScheme_2010 .
     }
}
```

Listing 7.8: SPARQL query to filter the CSSA1 data set (Source: author)

of the described data set. The values of the reference period represent even time intervals so the commensurability is assumed.

One way to solve is to slice a preprocessed cube having disproportionate dimension values into a set of smaller subcubes, in which the dimension values belong to the same level of a concept hierarchy. Measured values can be then discretized into intervals in each subcube separately. Number of subcubes that the main cube has to be divided into depends on the number dimensions and the number of levels in each dimension's hierarchies. Also when the commensurability cannot be expected among dimension values on the same level of their hierarchy, it is a good idea to *make a cut* for each dimension value. For example, the dataset `cssa-d:vydaje-na-duchody-v-cr`[27] capturing costs on pensions in the Czech Republic by year and kind of pension contains 10 distinct values of the dimension of pension kind (not considering the scheme used only for year 2008). The cube would have to be divided into 10 subcubes for each value of the pension kind dimension. The CSSA1 data set would have to be divided into 222 subcubes. It has 37 pension kinds. Dimension of sex has two hierarchy levels: each sex and total. The dimension of reference area is considered to have 3 hierarchy levels: State's total, regions and Prague districts combined with the regional districts since they are comparable in number of inhabitants.

$$37 \times 2 \times 3 = 222 \ \textit{subcubes}$$

The construction of a subcube from a *master* cube can be performed by a SPARQL CON-

[27] https://data.cssz.cz/resource/dataset/vydaje-na-duchody-v-cr

STRUCT query. An example of such query is shown in the listing 7.9. This query filters the triples of the CZSO data cube **czso-job-applicants-and-unemployment-rate** to create a smaller cube of statistics about job applicants and unemployment rate for districts by sex. Notice how the reference area values corresponding to districts are distinguished. After the reference area values are linked (see 7.7) to their CSSA counterparts, the ontology provided with the CSSA data sets can be reused.

```
PREFIX qb: <http://purl.org/linked-data/cube#>
PREFIX sdmx-c: <http://purl.org/linked-data/sdmx/2009/code#>
PREFIX czso: <http://data.czso.cz/ontology/>
PREFIX czso-rd: <http://data.czso.cz/resource/dataset/>
PREFIX cssa-rd: <https://data.cssz.cz/resource/dataset/>
PREFIX cssa-or: <https://data.cssz.cz/ontology/ruian/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

CONSTRUCT { ?observation ?p ?o }
WHERE {
    GRAPH czso-rd:job-applicants-and-unemployment-rate {
        ?observation qb:dataSet czso-rd:job-applicants-and-unemployment-rate ;
            ?p ?o ;
            czso:refArea ?refAreaCZSO .
        NOT EXISTS {
            ?observation czso:sex sdmx-c:sex-T .
        }
     }
    ?refAreaCSSZ owl:sameAs   ?refAreaCZSO.
    GRAPH cssa-rd:pomocne-ciselniky { ?refAreaCSSZ a cssa-or:Okres }
}
```

Listing 7.9: SPARQL query to create a subcube (Source: author)

For every subcube a similar query has to be created and performed over the master cube. For a cube that has to be divided into a small number of subcubes it is plausible to write (and save it for the documentation and repeatability purposes) and perform these queries manually. But there are cubes for which this would involve an hours long work. At the same time, it is an trivial activity that can easily be automated. For this preproccesing step for the CSSA1 dataset, a Scala script was written that creates 222 distinct SPARQL queries that construct 222 subcubes, saves each query to a text file and also creates a shell script that triggers all queries and saves a result of each query to a distinct file in the turtle format. The script is listed in A.1. This, however, still requires writing such script for each preprocced data cube and solve the problem of a time consuming resolution of the queries.

7.7 Linking

In order to find rules describing relations across multiple sources (meaning data cubes of CZSO, data cubes of CSSA and Wikidata triples) the entities either have to be assigned the same URIs or to be connected by the `owl:sameAs` statements. The shared dimensions of the CSSA and CZSO data cubes are the reference area, reference period and sex. The dimension values URIs used for these dimensions differ not only institution from institution but also data

cube from data cube from the same institution (In the CSSA data cubes, reference period is represented by an entity of a year and of the last day of the year as well). Linking of equivalent dimension values of the three dimensions is done by creating `owl:sameAs` statements.

7.7.1 Sex Dimension Values

It was already mentioned that the CSSA data cubes use proxy entities linking to the SDMX representations of sexes, whereas the CZSO data cubes use these representations directly. So the linking statements are already provided with the CSSA code list file. To extract these very triples, the query listed below can be used. These triples can be than loaded into a mining task involving mining from data cubes contain the dimension of sex instead of loading the whole code list file.

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>

CONSTRUCT { ?cssaSex owl:sameAs ?sdmxSex }
WHERE {
    GRAPH <https://data.cssz.cz/resource/dataset/pomocne-ciselniky> {
        ?cssaSex a <https://data.cssz.cz/ontology/sdmx/code/Sex> ; owl:sameAs ?sdmxSex
    }
}
```

Listing 7.10: Linking the sex dimension values

7.7.2 Reference Periods

In CSSA data cubes, values from two concept schemes are used for representing the year intervals: a years scheme and a days scheme. The entities in the schemes are proxy entitities linking to the `data.gov.uk` Time Intervals ontology. CZSO data cubes use the ontology's day scheme concepts directly. That means that every year is represented by three distinct URIs in the data cubes so two `owl:sameAs` statements are required for each year. A query was written that generates theses statements for every year entity in the CSSA code list:

7.7.3 Reference Areas

The proxy entities of the reference area in CSSA data set used to link to the same entities that are used by the CZSO data sets. This linking is no longer present in the CSSA's code list but can be retrieved from the Opendata.cz's SPARQL endpoint. The query listed below returns 114 linking triples for all districts (including the Prague district entity), all regions (including Prague), Prague districts and the entity of the whole state. The linking with the Wikidata's entities had to be performed manually.

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

CONSTRUCT {
    ?cssaYear owl:sameAs  ?cssaDay .
    ?cssaYear owl:sameAs ?dataGovDay .
    ?cssaDay owl:sameAs ?dataGovDay .
}
WHERE {
    GRAPH <https://data.cssz.cz/resource/dataset/pomocne-ciselniky> {
        ?cssaYear skos:inScheme <https://data.cssz.cz/ontology/years/YearsScheme>
        BIND (REPLACE(str(?cssaYear),".*(\\d{4})","$1") as ?cssaYearValue)
        ?cssaDay skos:inScheme <https://data.cssz.cz/ontology/days/DaysScheme>
        FILTER (REGEX(str(?cssaDay), ".*day.*12-31"))
        BIND (REPLACE(str(?cssaDay),".*(\\d{4})-12-31","$1") as ?cssaDayValue)
        ?cssaDay owl:sameAs ?dataGovDay
        FILTER (?cssaYearValue = ?cssaDayValue)
    }
}
```

Listing 7.11: Linking the reference periods

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>

CONSTRUCT {
    ?cssaArea owl:sameAs ?odArea
}
WHERE {
    ?cssaArea a ?class ; owl:sameAs ?odArea .
    FILTER (?class IN (
        <https://data.cssz.cz/ontology/ruian/Okres>,
        <https://data.cssz.cz/ontology/ruian/Vusc>,
        <https://data.cssz.cz/ontology/ruian/SpravniObvod>,
        <https://data.cssz.cz/ontology/ruian/Stat>
        )
    )
}
```

Listing 7.12: Linking the reference periods

7.8 Discretization

The RDFRules implementation provides discretization functionality. The discretization tasks are, howerever, federated to the implemented discretization algorithms of the EasyMiner-Discretization library[28]. The equifrequent and the equisize discretization were chosen for the purposes of this experiment. The count of intervals that is set to be created from the set of measured values while performing the equifrequent discretization determines how many a which rules are generated by the RDFRules algorithm. If the the values are discretized into a small number of intervals, more rules with should be geenerated but the measure values become coarse and information is lost. If creating too many intervals, more specific rules should be found but they happen to have lower support. When performing the equisize

[28]https://github.com/KIZI/EasyMiner-Discretization

discretization, coarser rules are found for the intervals created for a higher support.

To avoid guessing, which number of intervals and which minimal support suits best the preprocessed data, multiple discretizations were performed with different parameters for both discretization algorithms. The minimal support thresholds were calculated as absolute numbers of various percentages of observations in the discretized cubes. As it was already mentioned, the preprocessed cube has to be cut into subcubes with commensurable observations, measures in theses subcube have to be discretized separately and only after that the triples can be merged and performed the mining tasks on.

That means that the number of overall measurements multiplies by the number of distinct discretizations. A situation has to be avoided, when the instantiations of variable representing observations are involving observations not only from various subcubes. The approach to solve this problem no matter how many measures are assigned to each observation will be shown on a sample of data below.

```
@prefix qb: <http://purl.org/linked-data/cube#> .

<o1> qb:dataSet <original-dataset> ;
    <dimension1> <dimension1value1> ; <dimension2> <dimension2value1> ;
    <measure1> 25000 ;
    <measure2> 3 .

<o2> qb:dataSet <original-dataset> ;
    <dimension1> <dimension1value2> ; <dimension2> <dimension2value2> ;
    <measure1> 10000 ;
    <measure2> 10 .
```

Listing 7.13: Observation example

Each application of a discretization algorithm will create a new measurement triple for each measure and observation with an object of the assigned interval based on the discretization algorithm and the parameter. The objects in triples assigning the observations to a data set will be changed to point to the particular subcube. In the example below two discretizations for each measure were performed on the two observations. In the example the same pair of discretizations were performed on the two distinct measures, but that does not have to be so. Assigning multiple triples of the same measure is fine as far as the measure is differently discretized.

The discretized subcubes can be than merged into a singe data set a be performed mining tasks on. In each rule it has to be ensured, that the set of observations is limited to a certain subcube. For each cube in a rule the body of a rule has to contain an atom of a pattern `(?o qb:dataSet AnyConstant)`.

An alternative to creating subcubes based on the concept hierarchy in the master cube's dimensions is to use an unsupervised clustering algorithm (eg. k-means) to divide the observations into subcubes based on the proximity of their measures. After that the workflow is the same, the measures are discretized in the generated subcubes separately and than the

```
@prefix qb: <http://purl.org/linked-data/cube#> .

<o1> qb:dataSet <subcube1> ;
    <dimension1> <dimension1value1> ; <dimension2> <dimension2value1> ;
    <measure1> <subcube1_ef3_measure1_3>, <subcube1_es10_measure1_2> ;
    <measure2> <subcube1_ef3_measure2_2>, <subcube1_es10_measure2_1> .

<o2> qb:dataSet <subcube2> ;
    <dimension1> <dimension1value2> ; <dimension2> <dimension2value2> ;
    <measure1> <subcube2_ef3_measure1_3> , <subcube2_es10_measure1_2> ;
    <measure2> <subcube2_ef3_measure2_1> ,<subcube2_es10_measure2_1> .
```

Listing 7.14: Discretization example

subcubes are merged. But this brings two problems when used for the RDFRules algorithm.

1. There is no clear interpretation of the generated subcubes, because their observation can belong to different levels of the concept hierarchy in a dimension. Unlike with the previous approach where a generated subcube could be described as for example *Population in districts by age category*.
2. For each distinct measure in the master cube a set of subcubes would have to be generated. So if the observations are assigned multiple measures (as the CZSO observations are) the number of observations would multiple with the number of measures and the observation URIs would have to be distinguished as one observation cannot be assigned to multiple data sets by the `qb:dataSet` triple.

7.9 Mining Tasks

The source codes of the performed experiment are available in repository on github[29] in the directory *notebooks* in a form of jupyter[30] notebooks. A Scala kernel for jupyter (eg. *almond.sh*[31]) has to be install in order to execute the code directly from the notebooks (see *README.md*). Some of the notebooks are dedicated to preproccesing of a particular data set and other contain the performed mining tasks. In the preproccesing notebooks the preprocced data set is exported to a text file in the *Turtle* format and also saved into a cache file from which the data set is loaded into memory before a mining task. Github strictly enforces 100 MB size limit for each file so the cache files and *Turtle* files are not pushed into the repository. In order to run the mining tasks locally one has to create the cache files by running the code in the preproccesing notebooks.

[29] https://github.com/nvkp/diploma-thesis-code
[30] https://jupyter.org/
[31] http://almond.sh/

7.9.1 Relation between the pension expenses and the policital alignment of the state's government

A relation can be assumed between the expenditure on pensions in a state social security system and the political ideology of the state. A left leaning governments usually tend to spend more on social welfare than the right-wing governments. In this section it is shown how this relation can be described by association rules in the spirit of 4.1 mined from the data published by CSSA and Wikidata. Data from various countries would be more appropriate for mining such relation. The available data are, however, sufficient for a simple demonstration.

The data cube used[32] is published by CSSA and contains records of total annual expenses on particular pensions. From now on it will be denoted as *expenses*. The only measure of the data cube is the expendature amount in CZK. The dimensions of the data cube are the reference period with the granularity of whole years (from 2008 till 2019) and the dimension of the pension kind containing individual pension kind and also aggregated dimension values. For the year 2008 and 2009 and old pension scheme is used so the observations considering these two years were discarded. For the dimension of reference period 10 distinct values remain and the newer pension kind scheme contains also 10 distinct values. That would suggest that 100 observations remain but there are only 94. For the one-off allowance to pensions there are only 4 observations. That probably means that the expenses for this pension kind in the missing years were zero. So the missing observations with the measured value of zero were created manually.

The original data cube was sliced into 10 subcubes (by SPARQL queries) each containing 10 observations. Each subcube is in a separate file in the folder `data/expenses`. In the notebook `expenses.ipynb` each subcube's measures are discretized equifrequently (with 2 and 5 intervals) and equisizeably (with the relative support of 20, 30 and 50 percent) assigning each measurement 5 intervals.

The preproccesing of the Wikidata data set (notebook `wikidata.ipynb`) rested in merging the triples returned by queries A.2 and A.3 and removing the triples with predicates `P580` and `P582` as they only served to generate annual triples in the query A.3.

In the notebook (`expenses-wikidata.ipynb`) of the mining task the two preprocessed data sets were loaded and merged together with the reference period linking triples. From this data set containing 25 494 triples an object of index was created. The rules that should be found are described in plain text as *If in any year the current prime minister belongs to a political party that has a certain political alignment then the annual expenses of a certain pension kind fit into certain interval.* Let's write it as a *pseudo* rule pattern in the context of the available data's structure:

$$(\langle Czech\_Republic\rangle\ headOfGovernment\ ?headRole) \wedge (?headRole\ person\ ?person)$$

$$\wedge(?person\ partyRole\ ?partyRole) \wedge (?partyRole\ party\ ?party)$$

[32] https://data.cssz.cz/web/otevrena-data/-/vydaje-na-duchody-v-cr

$$\wedge(?party\ alignment\ AnyConstant) \wedge (?partyRole\ appliesTo\ ?refPeriod)$$

$$\wedge(?headRole\ appliesTo\ ?refPeriod) \wedge (?observation\ refPeriod\ ?refPeriod)$$

$$\wedge(?observation\ dataSet\ AnyConstant) \Rightarrow (?observation\ expenses\ AnyConstant)$$

The variable ?*headRole* would bind to binding entitities representing an acting of a specific person in the office just as the variable ?*partyRole* would bind to a binding entitites of memberships of a specific person in a specific political party. The rules yielded from this rule pattern would differ only by the entities at the object positions of the fifth and nineth atom and the head atom. The last body atom with the `qb:dataSet` will ensure that all ?*observation* bindings belong to the same sub cube and thus the commensurability of measures will be preserved. There is no need of an atom stating the pension kind, because all observations in a sub cube share the same one. That way the rule is shorter.

Human tends to write a rule pattern and understand a rule from left to right, from the body to the head, but the algorithm creates the rules in the opposite direction. When writing a rule pattern this fact has to be taken into consideration together with which refining operators are available to the algorithm. The rule pattern above makes perfect sense but the algorithm is not able to generate such rules. Have a look at the sixth body atom. When read from the head, the atom introduces new variable ?*partyRole* while the ?*headRole* variable is still open. The algorithm cannot add a new variable to a rule without closing an open variable in the rule. For this particular rule pattern the problem will be solved by moving the seventh atom to the first position in the body and by switching the positions of the fourth and fifth atom. Also the used algorithm implementation does not allow custom names for variables in a pattern. The variable names have to be single characters ordered alphabeticaly beginning with ?*a* from the rule's head. The definition of the rule pattern object to match the seeked relations in this mining task is shown in listing 7.15:

```
val alignmentExpenses: RulePattern = (
    AtomPattern(subject = 'f', predicate = appliesTo, `object` = 'b') &:
    AtomPattern(subject = czURI, predicate = wdProperty(6), `object` = 'f') &:
    AtomPattern(subject = 'f', predicate = wdProperty(6), `object` = 'e') &:
    AtomPattern(subject = 'e', predicate = wdProperty(102), `object` = 'c') &:
    AtomPattern(subject = 'd', predicate = wdProperty(1387)) &:
    AtomPattern(subject = 'c', predicate = wdProperty(102), `object` = 'd') &:
    AtomPattern(subject = 'c', predicate = appliesTo, `object` = 'b') &:
    AtomPattern(subject = 'a', predicate = cssaRefPeriod, `object` = 'b') &:
    AtomPattern(subject = 'a', predicate = qbdPredicate, `object` = AnyConstant)
    =>:
    AtomPattern(subject = 'a', predicate = expenses)
)
```

Listing 7.15: First pattern definition

This pattern was connected to the mining task definition itself. No minimal support an head coverage thresholds were set so that a maximal number of rules that can be filtered later are found. The maximal rule length set corresponds to the provided rule pattern. The mining

task generated 192 rules[33]. The rules with a support of 1 were filtered out and the PCA confidence, the standard confidence and lift was computed on the remaining 116 rules[34]. The listing below shows two of them as they are written in the export file.

```
(?f prfx:appliesToRefPeriod ?b) ^ (wd:Q213 p:P6 ?f) ^ (?f p:P6 ?e) ^ (?e p:P102 ?c) ^
    (?d p:P1387 "centre-right") ^ (?c p:P102 ?d) ^ (?c prfx:appliesToRefPeriod ?b) ^
    (?a cssa-od:refPeriod ?b) ^ (?a qb:dataSet <expenses-old-age-total>) -> (?a cssa-
    om:vydaje-na-duchody-opravene-o-zalohy-v-tis-kc <<3.1797095155E8__3.8222294828E8)
    _ef3_3/3>) | support: 3, headCoverage: 0.005, confidence: 1.0, pcaConfidence: 1.0,
     lift: 25.0, headConfidence: 0.04, headSize: 600, bodySize: 3, pcaBodySize: 3

(?f prfx:appliesToRefPeriod ?b) ^ (wd:Q213 p:P6 ?f) ^ (?f p:P6 ?e) ^ (?e p:P102 ?c) ^
    (?d p:P1387 "centrism") ^ (?c p:P102 ?d) ^ (?c prfx:appliesToRefPeriod ?b) ^ (?a
    cssa-od:refPeriod ?b) ^ (?a qb:dataSet <expenses-it>) -> (?a cssa-om:vydaje-na-
    duchody-opravene-o-zalohy-v-tis-kc <<2.488316469E7__2.553204263E7)_ef3_1/3>) |
    support: 2, headCoverage: 0.0033333333333333335, confidence: 0.6666666666666666,
    pcaConfidence: 0.6666666666666666, lift: 22.22222222222222, headConfidence: 0.03,
    headSize: 600, bodySize: 3, pcaBodySize: 3
```

Listing 7.16: First pattern's rule sample

The first rule states that when the government has the center-right political alignment, the total expenses for all types of old age pensions are in the upper third. The second rule predicts the lower third value of expenses on the third degree invalidity pension, when the government's policital alignment is centristic. Note that the head size values are distorted by the performed discretizations. Each of the 100 observations appears in the head size as many times as there are measure intervals assigned to the head measure predicate. This also distorts the head coverage. Also the lift is over estimated because its denominator works with all 100 observations when the only observations that matter are that of the sub cube specified in the rule's body.

Quite suprisingly in none of the 116 rules the left alignment is mentioned even though Czech Republic's prime minister from the years 2014 to 2017 was a left leaning politician. Only alignments mentioned are *centre-right* and *centrism*. The only governing party assigned to those values in the covered period is *ANO 2011*[35]. It seems that the party itself is a sufficient *explanatory variable* for the measure in the data cube. So the pattern was simplified to connect the measured values only to the appropriate party.

58 rules matching the pattern in 7.17 exceed support of 1. Only party appearing in these rule is the above mentioned *ANO 2011*. It is the latest governing party in the covered period so it could seem that, more than the relation between the pensions expendature and governing party, the rules describe a relation between pensions expendature and time because the measured values are not treated for inflation. There are, however, also rules that predict the lowest of the intervals.

[33] https://nvkp.github.io/diploma-thesis-code/rulesets/alignmentExpensesTaskRuleset.txt
[34] https://nvkp.github.io/diploma-thesis-code/rulesets/alignmentExpensesTaskRulesetFiltered.txt
[35] https://www.wikidata.org/wiki/Q10728124

```
val partyExpenses: RulePattern = (
    AtomPattern(subject = 'e', predicate = appliesTo, `object` = 'b') &:
    AtomPattern(subject = czURI, predicate = wdProperty(6), `object` = 'e') &:
    AtomPattern(subject = 'e', predicate = wdProperty(6), `object` = 'd') &:
    AtomPattern(subject = 'd', predicate = wdProperty(102), `object` = 'c') &:
    AtomPattern(subject = 'c', predicate = wdProperty(102), `object` = AnyConstant) &:
    AtomPattern(subject = 'c', predicate = appliesTo, `object` = 'b') &:
    AtomPattern(subject = 'a', predicate = cssaRefPeriod, `object` = 'b') &:
    AtomPattern(subject = 'a', predicate = qbdPredicate, `object` = AnyConstant)
    =>:
    AtomPattern(subject = 'a', predicate = expenses)
)
```

Listing 7.17: Second pattern definition

```
(?e prfx:appliesToRefPeriod ?b) ^ (wd:Q213 p:P6 ?e) ^ (?e p:P6 ?d) ^ (?d p:P102 ?c) ^
    (?c p:P102 wd:Q10728124) ^ (?c prfx:appliesToRefPeriod ?b) ^ (?a cssa-od:refPeriod
     ?b) ^ (?a qb:dataSet <expenses-old-age-total>) -> (?a cssa-om:vydaje-na-duchody-
    opravene-o-zalohy-v-tis-kc <<3.1797095155E8__3.8222294828E8)_ef3_3/3>) | support:
    3, headCoverage: 0.005, confidence: 1.0, pcaConfidence: 1.0, lift: 25.0,
    headConfidence: 0.04, headSize: 600, bodySize: 3, pcaBodySize: 3

(?e prfx:appliesToRefPeriod ?b) ^ (wd:Q213 p:P6 ?e) ^ (?e p:P6 ?d) ^ (?d p:P102 ?c) ^
    (?c p:P102 wd:Q10728124) ^ (?c prfx:appliesToRefPeriod ?b) ^ (?a cssa-od:refPeriod
     ?b) ^ (?a qb:dataSet <expenses-it>) -> (?a cssa-om:vydaje-na-duchody-opravene-o-
    zalohy-v-tis-kc <<2.488316469E7__2.553204263E7)_ef3_1/3>) | support: 2,
    headCoverage: 0.0033333333333333335, confidence: 0.6666666666666666, pcaConfidence
    : 0.6666666666666666, lift: 22.22222222222222, headConfidence: 0.03, headSize:
    600, bodySize: 3, pcaBodySize: 3
```

Listing 7.18: Second pattern's rule sample

7.9.2 Appending the YAGO Data Set to the CZSO Data Cubes

In this task the mining were performed over the CZSO's data cube *Job Applicants and Unemployment Rate*, from now on denoted as *JAUR*, and the triples extracted from YAGO data set. The data cube contains dimensions of reference area (89 distinct dimension values constisting of 13 regions without Prague and 76 districts without Prague, meaning there are no data concerning Prague), reference period (years 2005 to 2013) and sex (male, female and total). Each observation in the cube contains multiple measures. All observations are assigned the measures of number of available job applicants (job applicants who have no objective obstacle to taking up a job, eg enrollment in retraining courses or serving a sentence) and unemployment rate. Observations concerning both sexes also contain the measures of all job applicants and number of vacancies.

The data cube had to be sliced along the dimensions of reference area and sex. The dimension of reference area was divided into regions and district. It was considered to divide the dimension of sex by each dimension value, but there is no significant difference between the male and female values[36], so the dimension was divided into *total* and *by-sex* part. That

[36] https://nvkp.github.io/diploma-thesis-code/data/jaur/SexDimension

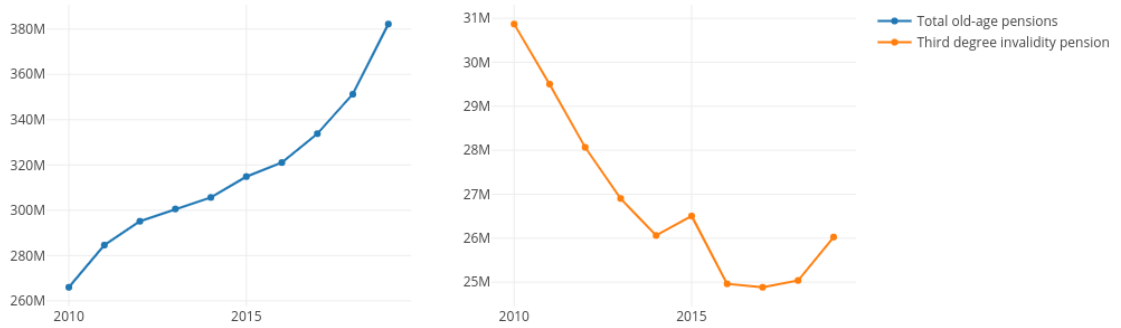

Figure 7.1: Expenditure of selected pension kinds over time

makes 4 slices [37] with 2 403 observations ($3 \times 9 \times 89$) and 6 408 measured values in total. All the measured values were discretized[38] equifrequently with the interval counts of 10, 30, 50 and 100 and equisizeably with the percentages of 1, 2 and 3. The notebook `jaur-yago`[39] contains the mining itself. The YAGO triples and data cube's triples were merged together with the appropriate reference area linking triples[40].

A separate index (containing all the extracted YAGO triples, the reference area linking triples and only those triples from the *JAUR* cube that belong to the subcube's observations) was created for each subcube. That way the lift measure calculated for the rules mined from the indexes is not distorted because its denominator is not increased by inappropriate observations.

Just as in the previous example, a pattern had to be provided to the mining tasks, that generates rules assigning a particular interval of a measure for observations satisfying the rule's body. In this pattern a variable at the position of object with the predicate of `refArea` appears in an atom that is enforced to bind to triples from the YAGO data set. For the mining task working with this pattern the threshold of maximal rule's length can be declared that corresponds to the number of hops of the extracted YAGO triples (determining the maximal length of the atom `chain` from YAGO) and the number of other atoms in the body and the head atom.

For fininishing the mining task in a reasonable time also a minimal support thresholds has to be declared. The support of the seeked rules corresponds to the number of observations for which the rule is valid. The rules have to contain an atom achoring the observations' variable to a constant of a slice by the `qb:dataSet` predicate to ensure that each rule relates to one and only slice. The slices of the *JAUR* data cube contain a different number of observations,

[37] https://nvkp.github.io/diploma-thesis-code/data/jaur/slice-queries/
[38] https://nvkp.github.io/diploma-thesis-code/notebooks/jaur
[39] https://nvkp.github.io/diploma-thesis-code/notebooks/jaur-yago
[40] https://nvkp.github.io/diploma-thesis-code/data/linking/yagoCZSOLinking.ttl

so if a too high minimal support threshold is declared, the potentionally accurate rules from the a smaller slice would be discarded. In 7.9.1 all the slices contain the same number of observations and the whole cube so small that no minimal support threshold did not have to be used. In this mining task the problem was avoided by creating 4 distinct patterns that anchor the observations to each slice. These patterns are then appended to a separate mining task with different minimal support thresholds.

```
val districtBySexPattern = (
    AtomPattern(subject='b',graph=uri("yago"))&:
    AtomPattern(subject='a',predicate=refArea,`object`='b',graph=uri("czso"))&:
    AtomPattern(subject='a',predicate=qbDataSet,`object`=districtBySexSlice,graph=uri(
        "czso"))
    =>:
    AtomPattern(subject='a',predicate=oneOfMeasures, graph = uri("czso"))
)
```

Listing 7.19: Pattern Definition for the Slice *districts-by-sex*

The smallest slice is `jaur-regions-total` with only 117 observations (9 years of 13 regions). Let's assume there is a rule that can be infered from this data that predicts a certain measure value to be in a certain interval given a certain characteristic of the reference area of the observation. Not to describe only one specific region, there have to be at least two regions satisfying the rule. If the rule had the confidence of 1 its support would be 18 (9 years $\times$ 2 regions). A rule with confidence of 0,5 would have half the support. Less confident rules with support of 9 or higher support would *cover* more regions. A minimal support for each mining task corresponding to single slice can be therefore given as the number of observations in the slice divided by number of observations for a distinct reference area. For the district slices such designated threshold would however not restrict the search space to finish the procedure in a reasonable time so for these two mining tasks with largest number of observations the number was multiplied by 3 (We demand *half confident* rules to concern at least 6 districts).

```
val districtBySexTask = Amie()
    .addThreshold(Threshold.MinSupport(minSupport(districtBySexSlice)*3))
    .addThreshold(Threshold.MaxRuleLength(6))
    .addConstraint(constantsOnlyAtObject)
    .addPattern(districtBySexPattern)
```

Listing 7.20: Mining Task Definition for the Slice *districts-by-sex*

RDFRules API provides a constraint that can be appended to the mining task that eliminates atoms with a constant at the position of subject to be considered during the rule refinement. This shrinks the task's search space and contributes to shorter runtime when such atoms are not relevant for the particular pattern. In 7.9.1 this constraint could not be used since the second atom pattern's subject in the rule pattern was the constant of the Czech Republic's URI. It makes sense to add the constraint now. It makes sense to add the constraint now.

Folows the description of the processing of each mining task's retrieved rule set. The used patterns do not prescribe a structure for each atom in a rule. The length of the rules can

vary from 4 to 6 atoms. The patterns do not prevent the refining operators from appending an atom to the rule's body, whose subject variable represents a different observation than the one in the rule's head. That practically introduces a new *unclosed* cube to the rule and invalidates it.

```
(?c czso:refPeriod <http://reference.data.gov.uk/id/gregorian-day/2006-12-31>) ^ (?c
    czso:refArea ?b) ^ (?b <http://schema.org/foundingDate> "2000-11-12") ^ (?a czso:
    refArea ?b) ^ (?a qb:dataSet <jaur-regions-total>) -> (?a czso:podilNezamestnanych
     <<9.01__11.47)_ef10_10/10>)
```

Listing 7.21: XXX

Atoms with a new observation variable in those rules are connected through the variable of reference area. They all contain atom with the new variable at the position of subject, reference area dimension URI at the position of predicate and the variable of reference area at the position of object. All those rules were filtered out. Only those rules in the rule set, that have exactly one atom with reference area dimension URI at the position of predicate were kept for further processing.

There is apparently a correlation between the cube's measures (available applicants is a subset of all job applicants, lower unemployment rate corresponds to lower number of vacancies). The algorithm could tend to create evident rules in the sense of *if there is a lower unemployment rate in the area, there is a lower number of applicants registered at the Labor Office.* Not to bother with this kind of rules, the rules with a measure URI at the position of predicate in any atom in the rule's body were filtered out as well. The lift was computed for the remaining rules in the rule sets. All rules the lift value lower than 1 were filtered out. The remaining rules were reduced by the *data coverage pruning* method.

TODO odkazy na výsledky [41]

TODO příklady pravidel, ukázat nějaké užitečné a ukázat nějaké vadné

7.9.3 Relation Between Measures from Different Data Cubes

7.10 Discussion

[41] https://nvkp.github.io/diploma-thesis-code/rulesets/jaur-yago/

Conclusions

List of References

[1] David Beckett et al. *RDF 1.1 Turtle*. 2014-02. URL: http://www.w3.org/TR/turtle/.

[2] Tim Berners-Lee. *Linked Data - Design Issues*. 2006. URL: http://www.w3.org/DesignIssues/LinkedData.html.

[3] Richard Cyganiak and Dave Reynolds. *The RDF Data Cube Vocabulary*. W3C Recommendation. https://www.w3.org/TR/2014/REC-vocab-data-cube-20140116/. W3C, 2014-01.

[4] Luis Galárraga et al. 'Fast rule mining in ontological knowledge bases with AMIE+'. In: *The VLDB Journal* 24 (2015-07). DOI: 10.1007/s00778-015-0394-1.

[5] Luis Antonio Galárraga et al. 'AMIE: Association rule mining under incomplete evidence in ontological knowledge bases'. In: 2013-05, pp. 413–422. DOI: 10.1145/2488388.2488425.

[6] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space*. Morgan & Claypool, 2011. ISBN: 9781608454303. URL: http://linkeddatabook.com/.

[7] Frank Manola and Eric Miller. *RDF Primer*. 2004-02. URL: http://www.w3c.org/TR/rdf-primer/.

[8] Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF*. 2008-01. URL: http://www.w3.org/TR/rdf-sparql-query/.

[9] Václav Zeman, Tomáš Kliegr, and Vojtěch Svátek. 'RDFRules Preview: Towards an Analytics Engine for Rule Mining in RDF Knowledge Graphs'. In: *RuleML Challenge* (2018).

[10] Václav Zeman, Tomáš Kliegr, and Vojtěch Svátek. 'RDFRules: Making RDF Rule Mining Easier and Even More Efficient'. In: *Semantic Web Journal* (2020). URL: http://www.semantic-web-journal.net/system/files/swj2511.pdf.

Appendices

A. Queries and Scripts

```
import scala.io.Source
import java.io.PrintWriter
import scala.sys.process._

val byRegion = """
?observation cssz-dimension:refArea ?refAreaCSSA .
GRAPH <https://data.cssz.cz/resource/dataset/pomocne-ciselniky> {
    ?refAreaCSSA a <https://data.cssz.cz/ontology/ruian/Vusc> .
}
"""

val byDistrict = """
?observation cssz-dimension:refArea ?refAreaCSSA .
GRAPH <https://data.cssz.cz/resource/dataset/pomocne-ciselniky> {
    ?refAreaCSSA a ?class .
    FILTER (?class IN (
        <https://data.cssz.cz/ontology/ruian/Okres>,
        <https://data.cssz.cz/ontology/ruian/SpravniObvod>
        )
    )
}
"""

val stateTotal = """
?observation cssz-dimension:refArea <https://data.cssz.cz/resource/ruian/staty/1> .
"""

val bySex = """
NOT EXISTS {
    ?observation cssz-dimension:pohlavi <https://data.cssz.cz/ontology/sdmx/code/sex-T
        >
}
"""

val bothSexes = """
?observation cssz-dimension:pohlavi <https://data.cssz.cz/ontology/sdmx/code/sex-T> .
"""

val template = """
PREFIX qb: <http://purl.org/linked-data/cube#>
PREFIX cssz-dimension: <https://data.cssz.cz/ontology/dimension/>

CONSTRUCT {
    ?observation ?p ?o
}
WHERE {

    ?observation qb:dataSet <https://data.cssz.cz/resource/dataset/duchodci-v-cr-
        krajich-okresech> .
    ?observation ?p ?o .
    %s
    ?observation cssz-dimension:druh-duchodu <%s> .
    %s
}
"""

val commandTemplate = "../../jena/bin/arq --data \"../../data/pensions-filtered.ttl\"
```

```
    --data \"../../data/pomocne-ciselniky.trig\" --query \"queries/%s.rq\" > \"results
    /%s.ttl\""
val source = Source.fromFile("pensionkinds.txt")
val lines = source.getLines().toArray
val pw = new PrintWriter(s"script.sh")
lines.foreach(line => {

    val kindName = line.replaceAll("https://data.cssz.cz/resource/pension-kind/","").
        replaceAll("_2010","")
    val districtTotalQuery = template.format(bothSexes,line,byDistrict)
    val districtTotalFileName = s"pensions-by-district-total-$kindName"
    val districtTotalPw = new PrintWriter(s"queries/$districtTotalFileName.rq")
    districtTotalPw.print(districtTotalQuery)
    districtTotalPw.close()
    pw.println(commandTemplate.format(districtTotalFileName,districtTotalFileName))

    val districtBySexQuery = template.format(bySex,line,byDistrict)
    val districtBySexFileName = s"pensions-by-district-by-sex-$kindName"
    val districtBySexPw = new PrintWriter(s"queries/$districtBySexFileName.rq")
    districtBySexPw.print(districtBySexQuery)
    districtBySexPw.close()
    pw.println(commandTemplate.format(districtBySexFileName,districtBySexFileName))

    val regionTotalQuery = template.format(bothSexes,line,byRegion)
    val regionTotalFileName = s"pensions-by-region-total-$kindName"
    val regionTotalPw = new PrintWriter(s"queries/$regionTotalFileName.rq")
    regionTotalPw.print(regionTotalQuery)
    regionTotalPw.close()
    pw.println(commandTemplate.format(regionTotalFileName,regionTotalFileName))

    val regionBySexQuery = template.format(bySex,line,byRegion)
    val regionBySexFileName = s"pensions-by-region-by-sex-$kindName"
    val regionBySexPw = new PrintWriter(s"queries/$regionBySexFileName.rq")
    regionBySexPw.print(regionBySexQuery)
    regionBySexPw.close()
    pw.println(commandTemplate.format(regionBySexFileName,regionBySexFileName))

    val totalTotalQuery = template.format(bothSexes,line,stateTotal)
    val totalTotalFileName = s"pensions-total-total-$kindName"
    val totalTotalPw = new PrintWriter(s"queries/$totalTotalFileName.rq")
    totalTotalPw.print(totalTotalQuery)
    totalTotalPw.close()
    pw.println(commandTemplate.format(totalTotalFileName,totalTotalFileName))

    val totalBySexQuery = template.format(bySex,line,stateTotal)
    val totalBySexFileName = s"pensions-total-by-sex-$kindName"
    val totalBySexPw = new PrintWriter(s"queries/$totalBySexFileName.rq")
    totalBySexPw.print(totalBySexQuery)
    totalBySexPw.close()
    pw.println(commandTemplate.format(totalBySexFileName,totalBySexFileName))
})
pw.close()
```

Listing A.1: Scala script for creating SPARQL queries for preproccesing the CSSA1 data set

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX p: <http://www.wikidata.org/prop/>

CONSTRUCT {
```

```
    ?area p:P31 wd:Q5153359.
    ?area p:P6 ?headRole ;
        p:P131 ?superiorArea .
    ?headRole p:P6 ?person ;
        p:P580 ?headRoleStartDate ;
        p:P582 ?headRoleEndDate .
    ?person p:P102 ?partyRole .
    ?partyRole p:P580 ?partyRoleStartDate ;
            p:P582 ?partyRoleEndDate ;
            p:P102 ?party .
    ?party p:P1387 ?alignmentLabel .
}
WHERE {
    {
        ?area wdt:P31 wd:Q38911 .
    } UNION {
        ?area wdt:P31|p:P31 wd:Q5153359 .
        OPTIONAL {
            ?area wdt:P131|p:P131 ?superiorArea
        }
    } UNION {
        wd:Q3342946 wdt:P1269 ?area
    }
    OPTIONAL {
        ?area p:P6|wdt:P6 ?headRole .
        ?headRole ps:P6 ?person ;
            pqv:P580|pq:P580 ?headRoleStartDate .
        OPTIONAL {
            ?headRole pqv:P582|pq:P582 ?headRoleEndDate .
        }
        OPTIONAL {
            ?person p:P102|wdt:102 ?partyRole .
            OPTIONAL {
                ?partyRole pqv:P580|pq:P580 ?partyRoleStartDate .
            }
            OPTIONAL {
                ?partyRole pqv:P582|pq:P582 ?partyRoleEndDate .
            }
            OPTIONAL {
                ?partyRole ps:P102 ?party .
                ?party wdt:P1387|p:P1387 ?alignment .
                ?alignment rdfs:label ?alignmentLabel
                FILTER (lang(?alignmentLabel) = "en")
            }
        }
    }
}
```

Listing A.2: Extracting the Wikidata's political data about the Czech Republic

```
PREFIX cssa-ody:  <https://data.cssz.cz/ontology/days/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX p: <http://www.wikidata.org/prop/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dt: <http://kizi.vse.cz/novp19/diploma-thesis/>

CONSTRUCT {
    ?headRole dt:appliesToRefPeriod ?refPeriod .
    ?partyRole dt:appliesToRefPeriod ?refPeriod .
}
WHERE {
```

```
    GRAPH <https://data.cssz.cz/resource/dataset/pomocne-ciselniky> {
        ?refPeriod skos:inScheme cssa-ody:DaysScheme .
        FILTER REGEX(str(?refPeriod), ".*year.*")
        FILTER REGEX(str(?refPeriod), "^.*\\d{4}-12-31")
        BIND(REPLACE(str(?refPeriod), "^.*(\\d{4}).*","$1") as ?refPeriodBind)
    }
    ?area p:P6 ?headRole .
    ?headRole p:P580 ?headRoleStartDate .
    FILTER ( datatype(?headRoleStartDate) = xsd:string)
    BIND(REPLACE(str(?headRoleStartDate), "^(\\d{4}).*","$1") as ?
        headRoleStartDateBind)
    OPTIONAL {
        ?headRole p:P582 ?headRoleEndDate .
        FILTER ( datatype(?headRoleEndDate) = xsd:string)
        BIND(REPLACE(str(?headRoleEndDate), "^(\\d{4}).*","$1") as ?
            headRoleEndDateBind)
    }
    FILTER (
        (bound(?headRoleEndDateBind) && ?refPeriodBind >= ?headRoleStartDateBind && ?
            refPeriodBind <= ?headRoleEndDateBind)
        ||
        (!bound(?headRoleEndDateBind) && ?refPeriodBind >= ?headRoleStartDateBind && ?
            headRoleStartDateBind > "2000")
    )
    OPTIONAL {
        ?headRole p:P6 ?person .
        ?person p:P102 ?partyRole .
        OPTIONAL {
            ?partyRole p:P580 ?partyRoleStartDate .
            FILTER (datatype(?partyRoleStartDate) = xsd:string)
            BIND(REPLACE(str(?partyRoleStartDate), "^(\\d{4}).*","$1") as ?
                partyRoleStartDateBind)
        }
        OPTIONAL {
            ?partyRole p:P582 ?partyRoleEndDate .
            FILTER ( datatype(?partyRoleEndDate) = xsd:string)
            BIND(REPLACE(str(?partyRoleEndDate), "^(\\d{4}).*","$1") as ?
                partyRoleEndDateBind)
        }
        FILTER (
            (bound(?partyRoleEndDateBind) && ?refPeriodBind >= ?partyRoleStartDateBind
                 && ?refPeriodBind <= ?partyRoleEndDateBind)
            ||
            (!bound(?partyRoleEndDateBind) && ?refPeriodBind >= ?
                partyRoleStartDateBind && ?partyRoleStartDateBind > "2000")
            ||
            (!bound(?partyRoleStartDateBind) && !bound(?partyRoleEndDateBind))
        )
    }
}
```

Listing A.3: Creating the **appliesToRefPeriod** triples