

Finding and sharing GIS methods based on the questions they answer

S. Scheider^a, A. Ballatore^b, and R. Lemmens^c

^aDepartment of Human Geography and Spatial Planning, Utrecht University, Utrecht, NL

^bDepartment of Geography, Birkbeck, University of London, London, UK

^cDepartment of Geoinformation Processing, ITC, University of Twente, Enschede, NL

ARTICLE HISTORY

Compiled April 26, 2018

Author copy, 2018

International Journal of Digital Earth

<https://doi.org/10.1080/17538947.2018.1470688>

ABSTRACT

Geographic information has become central for data scientists of many disciplines to put their analyses into a spatio-temporal perspective. However, just as the volume and variety of data sources on the Web grow, it becomes increasingly harder for analysts to be familiar with all the available geospatial tools, including toolboxes in Geographic Information Systems (GIS), R packages, and Python modules. Even though the semantics of the questions answered by these tools can be broadly shared, tools and data sources are still divided by syntax and platform-specific technicalities. It would, therefore, be hugely beneficial for information science if analysts could simply ask questions in generic and familiar terms to obtain the tools and data necessary to answer them. In this article, we systematically investigate the analytic questions that lie behind a range of common GIS tools, and we propose a semantic framework to match analytic questions and tools that are capable of answering them. To support the matching process, we define a tractable subset of SPARQL, the query language of the Semantic Web, and we propose and test an algorithm for computing query containment. We illustrate the identification of tools to answer user questions on a set of common user requests.

KEYWORDS

Question answering ; GIS methods ; SPARQL ; Semantic workflows ; Query containment

1. Introduction

More and more tools and methods for geospatial data analysis are being developed and distributed on the Web. Many analysts and researchers share their code as Python modules and as R packages (Müller et al., 2013). For this reason, the amount of available tools and data sources is becoming so large that a single analyst is hardly capable of keeping track of all of them. For example, in 2015, the number of R packages available on CRAN was 6,789, about 150 times as many commands available in commercial statistical packages such as SAS.¹ Even a single commercial GIS software suite such as ESRI's ArcGIS² contains hundreds of tools, and it is a challenge for analysts to understand and efficiently exploit their capabilities.

In recent years, several initiatives have been seeking to publish workflows as linked data (LD) on the Web (Belhajjame et al., 2015; Hofer et al., 2017). This should make it more easy for GIS analysts to search, find, and exchange *methods*, rather than just code and data (Scheider and Ballatore, 2018). As noted by many authors, the main advantage is that, while code is intrinsically bound to narrow technical specifications, methods are easily adaptable to new data, platforms, and contexts (Rey, 2009; Hinsén, 2014; Müller et al., 2013; Bernard et al.,

2014). However, this requires describing methods and related tools at a high level of abstraction. Early approaches to systematize GIS tools based on their analytic functionality failed, mainly because of the difficulty of abstracting from arbitrary details in their engineering and implementation (e.g., Albrecht, 1998).

The technical complexity of available tools tends to hide the fact that they often answer rather simple analytic questions, like wheels re-invented many times across platforms and communities. This issue is well understood by spatial information scientists, and a small set of core concepts can be identified as a possible abstraction layer for geospatial questions (Kuhn, 2012). The variety of questions that can be asked about such concepts is presumably limited, just as the variety of questions about factual knowledge on the Web.³

Unfortunately, even though a question is the driving force behind every analysis and is decisive for selecting both tools and data, current GIS are not capable of representing and handling questions in an explicit and machine readable way. GIS are incapable of letting analysts ask questions to find and employ those tools and data from the Web that would provide them with answers. For this reason, analysts are currently forced to formulate their questions in terms of the many awkward formats required by the analytic resources (Kuhn, 2012). This is an approach that neither leads to an inter-operable form of analysis nor does it scale with the variety of resources on the Web.

While there are many possible approaches to question-based interaction with analytic tools, from keyword matching (Gao and Goodchild, 2013), to service type matching (Zhao et al., 2012) and question answering (QA) (Lin, 2002), we suggest that question-based analysis in GIS needs to involve some *explicit representation of the question itself* at the level of generalizable spatial concepts. In this article, we investigate how common GIS tools can be captured in terms of the questions they answer using *SPARQL_{CONSTRAINT}*, a subset of the SPARQL⁴ query language for the Semantic Web. Using this language, we show how tools can be matched to requests by an algorithm for query containment, using ordinary Semantic Web reasoning on ontologies about interrogative spatial concepts. This approach supports the development of a platform-independent representation of tools, and allows analysts to identify tools based on the geospatial questions they answer. All the resources used in this article are available online.⁵

In the following section, we start with giving a quick review of computational approaches to question-based data analysis. In Section 3, we argue why Datalog based queries are not enough to capture GIS questions, and show how to turn questions into SPARQL queries in Section 4. In Section 5, we introduce *SPARQL_{CONSTRAINT}* and propose a corresponding procedure for determining whether a tool description matches a request formulated in this language. The approach is tested by requests on the example tools in Section 6, before we conclude and give an outlook on further research in Section 7.

2. Current approaches to question-based data analysis

Interacting with tools in terms of understandable questions has a large potential for future information technology. This is demonstrated, e.g., by Artificial Intelligence (AI) driven digital personal assistants, such as Apple’s Siri, Amazon’s Alexa, Microsoft’s Cortana or Google’s Home-Assistant (Canbek and Mutlu, 2016). The principle behind these assistants is that a user does not have to figure out the particular way how an app handles requests or data in order to make use of its service. He or she simply formulates a question such as “What is the weather like today?,” and the digital assistant invokes a weather web service that provides an answer to the question, feeding it with the necessary input, such as the location of interest, and delivering back an answer. However, digital assistants nowadays are incapable of figur-

ing out appropriate information services on their own. For example, in the case of Alexa, a weather app needs to be registered in terms of Alexa skills and triggered by explicitly stored keywords.

In information retrieval, question answering (QA) is seen as a possible paradigm to overcome the limitations of keyword-based querying (Allan et al., 2012). This approach seeks to automate answering questions about factual knowledge by querying over Web resources that potentially contain answers, based on named entity recognition or similarity of linguistic patterns (Lin, 2002). In the recent past, IBM’s achievement in Artificial Intelligence, a computer system winning *Jeopardy!* against human champions, was carried out in this way.⁶ These QA systems make use of the huge redundancy of answer formulations contained in Web documents, making it possible to match linguistic patterns present in a question. However, as was noted already by Lin (2002), a purely text-based approach fails with questions involving more complex concepts that require reasoning.

Along similar lines, matching of tools to questions based on keywords (Gao and Goodchild, 2013) is difficult because the concepts used to formulate questions and to describe answers may be on different semantic levels (Ofoghi et al., 2008). For example, in the context of air quality assessment, a raster GIS tool in ArcGIS, such as Raster Calculator, becomes an essential method for assessing the environmental influence on a person’s health (Kwan, 2016). Yet nothing in its name or the official tool description expresses this fact explicitly.⁷ Moreover, an analytic question abstracts not only from particular tool names or input formats, but also from particular solutions implemented in a tool. QA systems therefore also use semantic structures⁸ and make use of data cubes on the Semantic Web (Höffner et al., 2016; Mazzeo and Zaniolo, 2016).

A relevant research area aims at translating natural language questions into executable queries. Controlled natural languages (CNL) use parsers to bridge the gap between machine-readable queries and human-readable questions (Schwitter, 2010). In the Semantic Web, several attempts have been made to map the query language SPARQL⁹ and natural language (Rico et al., 2015; Ferré, 2014; Ngonga Ngomo et al., 2013). Using the Semantic Web as a platform for question-based interaction, interrogative concepts can, in principle, be shared and reused across the Web (Scheider and Lemmens, 2017). However, to really reap the benefits of this approach for question-based GIS, it is necessary to unpack and formalize these interrogative concepts in greater depth.

Another research area that has been addressing both query concepts and corresponding technology is that of service request matching. Geoweb service standards, such as OGC’s Web Processing Services (WPS), mainly rely on textual metadata (OGC, 2015), while researchers have proposed formal, ontology-based service descriptions,¹⁰ focusing on methods’ input, output, preconditions, and postconditions (Visser et al., 2002; Lemmens et al., 2006; Ludäscher et al., 2006; Lutz, 2007; Fitzner et al., 2011).

Describing tools based on the types of input and output is a common approach to make computational functions more findable (Albrecht, 1998). However, to effectively reuse GIS methods, types of input and output, also known as the data type signature, are not enough (Hofer et al., 2017). First, different methods can have the same signature and thus cannot be distinguished based on it. For example, the choropleth map classification method, available in ArcMap,¹¹ allows analysts to determine and visually compare the attribute class into which each region of a given spatial layer falls. However, it is not sufficient to know that the method takes a region layer as input and generates a map, because all mapping techniques essentially do this.

Second, it is necessary to capture complex logical relations, e.g., functional constraints, between inputs and outputs that are not easily expressed with data types (Fitzner et al., 2011), such as the ones underlying areal interpolation, where attribute values of regions are estimated

based on the values of overlapping regions. As we will articulate below, expressing such relationships requires a highly expressive interrogative language. Lutz (2007) and Fitzner et al. (2011) suggested to capture questions about the capability of a service in terms of queries, using Horn rules/Datalog with concepts taken from ontologies. Questions need to be formulated in terms of Datalog queries, and these queries need to be matched by determining whether a request query *contains* a service query. This task is well known as *query containment* in logic and database theory (Calì et al., 2012). While we follow this basic idea in this article, we will show in Section 3 why Datalog is not sufficiently expressive, and why we deem SPARQL to be a more appropriate language.

Recent approaches to make geooperators reusable with linked data on the Web (Brauner, 2015; Hofer et al., 2017) do not provide a systematic theory of the involved functionality. Other authors have attempted at describing methods in terms of spatial core concepts (Kuhn, 2012; Kuhn and Ballatore, 2015), and recently, in terms of usage patterns on the Web in a bottom-up manner (Ballatore et al., 2018). Unlike these authors, in what follows, we build on an approach to formalizing questions using SPARQL.

3. Why Datalog is not enough

What are appropriate strategies for capturing the functionality behind GIS tools? Datalog is a logic programming language based on Prolog that has been presented as a promising way of describing geoprocessing services (Fitzner et al., 2011). In this section, however, we show that Datalog is insufficient for capturing certain geospatial functionality. In what follows, we express free variables with a preceding question-mark $?x$. Datalog rules are of the form:

$$\text{rule } (body \Rightarrow head): \quad \forall x, \dots, z. \quad P_1(x, \dots, y) \wedge \dots \wedge P_n(w, \dots, z) \Rightarrow P_h(x, \dots, z) \quad (1)$$

$$\text{conjunctive query (no head):} \quad P_1(?x, \dots, ?y) \wedge \dots \wedge P_n(?w, \dots, ?z) \quad (2)$$

Note that variables x, \dots, z can be substituted by constants denoting instances, and P_i are predicates ranging over instances. When both requests (R) and methods (M) are represented as Datalog queries, then it becomes possible to match them in an efficient way based on query containment, i.e., testing whether the results of one query contain those of the other (Lemmens, 2006, Sect. 6.4) ($M \subseteq R$):

Definition 1. A query Q_1 is contained in a query Q_2 , written $Q_1 \subseteq Q_2$, if the set of facts obtained from Q_1 is a subset of facts obtained from Q_2 .

For example, if we request for an overlay operation with two spatial regions as inputs ($?x, ?y$) and one region as output ($?z$), then this strategy would return the intersection method, since intersection is subsumed by overlay, and thus all results returned by intersection are contained in the results returned by overlay (see also Lemmens, 2006, pp. 173):

$$\text{R query: } Region(?x) \wedge Region(?y) \wedge Region(?z) \wedge Overlay(?x, ?y, ?z) \quad (3)$$

$$\text{M query: } Region(?x) \wedge Region(?y) \wedge Region(?z) \wedge Intersect(?x, ?y, ?z) \quad (4)$$

The advantage of handling questions with query containment is that we do not have to know the answer (the query result) in order to know whether a method is useful for answering them. As can be seen above, however, Datalog has important syntactic restrictions (Calì et al., 2012), including:

- (1) Variables range only over instances, and never over predicates (= classes or relations)

- (2) Any variable in the head of a rule must also appear in the body (no existential quantification in the head, i.e., no expressions of the form $\forall. \Rightarrow \exists$, “for all ... there exists ...”)

While these restrictions make Datalog reasoning as well as query containment efficiently computable, they also implicate that important methods cannot be adequately described. Suppose we want to express choropleth classification:

$$hasAttr(?l, ?a) \wedge classOfScheme(?Class, ?s) \wedge ?Class(?a) \quad (5)$$

In other words, we are looking for the class of a given classification scheme that applies to the attribute value of a given region layer. However, this requires a variable $?Class$ that ranges over predicates, not instances, hence contradicting restriction 1. Furthermore, consider Areal Interpolation, which asks for a region layer $?l_{igt}$ in which all attribute values a were derived based on some interpolation operation o , being a parameter of the method, using layer $?l$:

$$\forall a. hasElmnt(?l_{igt}, ?e) \wedge hasAttr(?e, a) \Rightarrow \exists o. hasInp(o, ?l) \wedge hasOutp(o, a) \quad (6)$$

To describe this method, we have to quantify over the inner operation o , which is impossible in Datalog. For these reasons, we suggested in Scheider and Lemmens (2017) to use the more expressive SPARQL language.

4. Describing GIS questions as SPARQL queries

SPARQL 1.1 is the main query language of the Semantic Web.¹² In contrast to Datalog, it is based on Resource Description Framework (RDF), a logic with very expressive formal semantics, which is also the basis of linked data. In effect, RDF is a higher-order language (Hitzler et al., 2009). While Semantic Web reasoning languages such as OWL2 profiles¹³ and RDFS are first-order to stay within decidable bounds, SPARQL itself is not a language for reasoning, but only for querying a knowledge base containing explicit facts. There are three features of SPARQL/RDF that make it a suitable candidate for solving our method description problem:

- (1) As a higher-order language, it allows quantification over relations and classes. Relations are the predicates in linked data triples, i.e., the “arrows” that connect subjects to objects. Classes are objects of the predicate *rdf:type*, which is abbreviated as *a*. Both classes and predicates can be subject of further triples.

$$subject \xrightarrow{predicate} object \quad (7)$$

- (2) It allows *distinguishing bound and unbound variables* to tell goals (what you want to know) from other unknowns in a method. Bound variables are inside a SELECT or a CONSTRUCT clause.
- (3) It allows expressing *unrestricted negation* and *existential rules* (Mugnier and Thomazo, 2014) in terms of two nested FILTER NOT EXISTS statements:

$$FILTER NOT EXISTS\{ body \} FILTER NOT EXISTS\{ head \} \quad (8)$$

This corresponds to a logical statement of the form $\neg \exists (body \wedge \neg \exists . head)$, or equiva-

lently:

$$\forall .body \Rightarrow \exists .head \quad (9)$$

where both *body* and *head* are arbitrary graph patterns whose free variables are universally/existentially quantified, respectively. Such rules are needed to express questions with extrema, like “What is the attribute value of the nearest object?”, or to express quantified constraints over datasets, such as “A layer in which all attribute values were interpolated”. We will use this kind of rule statement extensively in the following.

In the remainder of this section, we suggest a selection of common GIS tools and formalize underlying questions as SPARQL queries.

4.1. GIS tools and informal questions

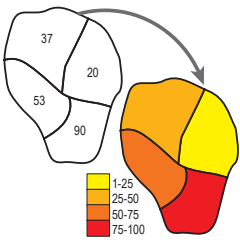
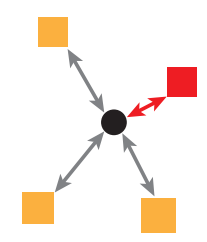
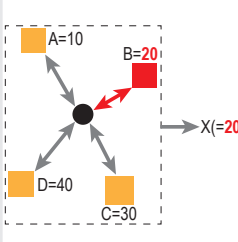
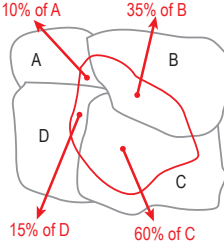
Most GISs include a recurring set of tools to perform operations on spatial data. Each tool can be thought of answering questions about the data. To develop our approach, we have selected a sample of GIS tools. As there is no broadly accepted hierarchy of GIS tools to draw upon for this selection, we have chosen a set of tools that are (1) conceptually diverse and non-overlapping, (2) include vector and raster operations, and (3) are well-known among GIS users. These tools are present in many commercial and open-source GIS packages,¹⁴ and despite having intuitively clear semantics, they embed complex details. The tools and their corresponding questions are summarized in Figure 1, and are formalized in Section 4.3.

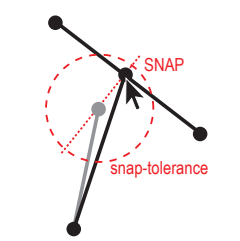
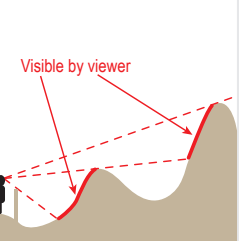
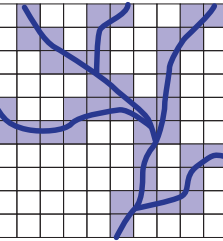
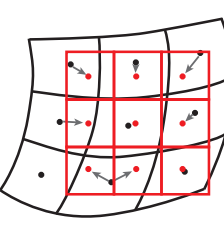
4.2. Interrogative vocabulary

One challenge of describing tools and their underlying questions across implementations is finding the right level of abstraction. Ontology design patterns, small reusable patterns of concepts, can help identifying the core concepts needed for this purpose (Janowicz, 2016). We reformulate analytical questions using the RDF vocabulary *AnalysisData*¹⁵ to represent datasets (layers) in terms of their data elements. A data element may link e.g. a single spatial region (called support) to some attribute value (called measure) – see Figure 2 and also Scheider and Ballatore (2018). Furthermore, we use well-known concepts from the *GeoSPARQL* ontology,¹⁶ *GeoSPARQL* functions,¹⁷ and properties for relational operators \leq and $=$.¹⁸ The principle behind this is to resolve all *FILTER* expressions, e.g. *FILTER*(*?a* = *?b*), into basic graph patterns (i.e., a set of triples). This simplifies the query matching process and further unifies SPARQL over implementations in different databases. Dataset-related concepts and functional GIS relations are summarized in Table 1. Note that these resources are available online.¹⁹

Besides simple relations, we also need to capture complex, n-ary relations and operations with RDF. For this purpose, we use the *Workflow* vocabulary,²⁰ which describes applications of operations in terms of their inputs and outputs (Scheider and Ballatore, 2018). Hence, we reify n-ary operation tuples in terms of nodes linking inputs to outputs. For example, *operation*(*a, b, c*) = *d* can be rewritten as a set of triples of the form: *operation wf:input ?a*, *operation wf:output ?d*, etc. These GIS operation nodes can be of a hierarchy of types, captured as *classes* such as in *GeoSPARQL* or the ontology *GISConcepts*.²¹ An example is *geof:distance*, which measures the distance between two geometries with respect to some unit of measurement (e.g. *xsd:meter*). n-ary relations are treated simply as boolean operations (with True/False output). An example is *gis:Visible* (see Table 2), which determines whether some location is visible from another given a height model (a layer). Note that the latter two operational

Figure 1.: Example tools and the informal questions they answer.

1 Choropleth classification	2 Nearest	3 NearTranspose	4 Area interpolation
ArcGIS	ArcGIS: Find Nearest	(forms part of ArcGIS/ILWIS tools)	ArcGIS: Area Interpolation Layer To Polygons
			
Given a layer I and a classification scheme s , in which attribute classes of s lie I 's regions?	Given layer I containing object o and a distance measure d , what is the nearest object to o ?	Given layer I with object o with attribute a , what is the attribute value of the nearest object?	Given layer I with regional aggregations and region layer I_{target} , which attribute values do the regions of I_{target} have when interpolating layer I ?

5 Snap	6 Viewshed	7 Vector to Raster conversion	8 Resample raster
ArcGIS: Snap tool	ArcGIS: Viewshed analysis	ILWIS: Polygon to raster	ILWIS: Resample
			
Given a set of geometries, how does the snapped geometry look like? The snapped version has every vertex within a threshold distance on the boundary of the target geometries.	Given a set of observer points and a digital elevation model (raster), which raster cells are visible from these points?	Given a polygon layer and a raster layer, which raster layer attributes can be derived from the polygon layer.	Given a raster layer as input and a target raster layer with different georeference, which attribute values in the target raster layer can be derived from the input layer.

Concepts	Usage	Description
<i>ada:hasElement</i>	ada:DataSet <i>ada:hasElement</i> ada:Data	Links datasets (e.g. layers) to their elements (data items)
<i>ada:hasMeasure</i>	ada:Data <i>ada:hasMeasure</i> ada:Reference	Links data items to their attribute values
<i>ada:hasSupport</i>	ada:Data <i>ada:hasSupport</i> ada:Reference	Links data items to their support values (e.g. a geometry)
<i>gis:RegionDataSet</i>	ada:DataSet a <i>gis:RegionDataSet</i>	A dataset with regions as supports
<i>gis:Vector</i>	ada:DataSet a <i>gis:Vector</i>	A vector data set
<i>gis:Raster</i>	ada:DataSet a <i>gis:Raster</i>	A raster data set
<i>geof:boundary</i>	geo:Geometry <i>geof:boundary</i> geo:Geometry	Links a boundary to a geometry
<i>geo:sfContains</i>	geo:Geometry <i>geo:sfContains</i> geo:Geometry	For example, lines contain points
<i>geo:sfEquals</i>	geo:Geometry <i>geo:sfEquals</i> geo:Geometry	Coinciding geometries
<i>m:leq</i>	8 <i>m:leq</i> 10	\leq (less than or equal)
<i>owl:sameAs</i>	8 <i>owl:sameAs</i> 8	$=$ (equality)

Table 1.: Classes and properties describing datasets and functional relations in GIS.

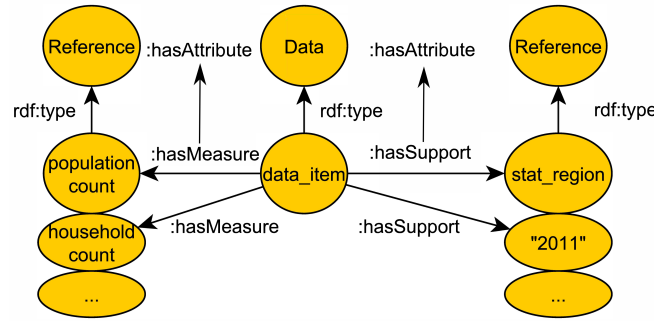


Figure 2.: Data items, supports and measures in the *AnalysisData* ontology.

types are treated as classes of operation nodes in RDF.

Given this vocabulary, on which ontological level should we describe tools and questions? Following the principle of query matching, we suggest that inside an ontology, tool descriptions should use concepts as concrete as possible to store enough detail, while interrogative concepts used for requests should be as general as possible, in order to maximize recall of tool queries.²²

We will describe GIS operations and their questions in the following in terms of CONSTRUCT queries. In the CONSTRUCT clause, we identify the operation and distinguish its inputs from outputs (see Figure 3), while in the WHERE clause, we formulate its inherent question. In this way, operational statements can be reused to define new questions. As shown below, this allows treating method queries as modules, simplifying question formulation and the matching process.

4.3. Translation of GIS operations to SPARQL

In this subsection, we show how GIS operations in Figure 1 can be described as SPARQL CONSTRUCT queries, relying on nested FILTER constructs.

Concepts	Usage	Explanation
<i>wf:fstInput</i> / <i>wf:sndInput</i>	<i>wf:Operation wf:fstInput</i> \top	Links operations to their first/second input
<i>gis:param</i>	<i>wf:Operation gis:param</i> \top	Links operations to a parameter as input
<i>wf:output</i> <i>gis:Visible</i>	<i>wf:Operation wf:output</i> \top <i>gis:Visible(a,b,c)</i>	Links operations to output <i>a</i> is visible from <i>b</i> with respect to height model <i>c</i>
<i>geof:distance</i>	<i>geof:distance(a,b,c) = d</i>	Distance from <i>a</i> to <i>b</i> in unit <i>c</i>

Table 2.: Concepts describing operations and complex relations in GIS.

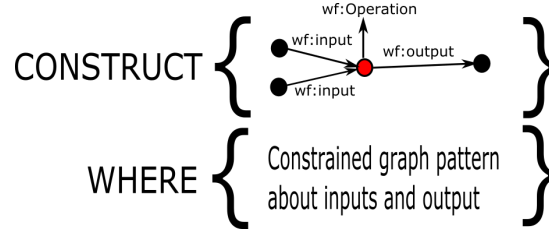


Figure 3.: Principle of capturing the semantics of an operation in terms of a construct query. The CONSTRUCT clause captures the operation signature, and the WHERE clause the underlying question.

Choropleth classification In the choropleth classification case (Figure 1.1), we query over the classes of a particular class scheme *?s_in*, given as parameter to the method, together with a region layer *?l_in* (Listing 1). Classes are linked via *ada:classOfScheme* to this scheme, and apply to the attribute values of *?l_in*. The output of this method are (a list of) pairs of data item *?e* with corresponding class *?class_out*.

```

CONSTRUCT {
  ?ch wf:fstInput ?l_in;
  gis:param ?s_in;
  wf:output ?class_out;
  wf:output ?e;
  a gis:ChoroClass.
} WHERE {
  ?l_in a gis:RegionDataSet;
  ada:hasElement ?e.
  ?e ada:hasMeasure ?attr.
  ?class_out ada:classOfScheme ?s_in.
  ?attr a ?class_out.
}

```

Listing 1: Choropleth classification

Nearest In this operation (Figure 1.2), we query for the object *?a_out* in a layer *?l_in* that is nearest to another given object *?b_in*. More precisely, we query for the object *?a_out* such that its distance (captured by another operation *geof:distance*) to *?b_in* is smaller than or equal to the distance to any other object within *?l_in*.

```

CONSTRUCT{
  ?n wf:output ?a_out;
  wf:fstInput ?b_in;
  wf:sndInput ?l_in;
}

```

```

    a gis:Nearest.
}WHERE{
  ?l_in ada:hasElement ?a_out.
  ?l_in a gis:Layer.
  ?a_out ada:hasSupport ?ar.
  ?b_in ada:hasSupport ?br.
  ?dist wf:fstInput ?ar.
  ?dist wf:sndInput ?br.
  ?dist a geof:distance.
  ?dist wf:output ?dv. # Distance of pair a,b
  FILTER NOT EXISTS{
    ?l_in ada:hasElement ?c.
    ?c ada:hasSupport ?cr.
    ?dc wf:fstInput ?cr.
    ?dc wf:sndInput ?br.
    ?dc a geof:distance.
    ?dc wf:output ?dcv. # Distance of pair c,b
  }
  FILTER NOT EXISTS{ ?dv m:leq ?dcv. }
}}

```

Listing 2: Nearest

NearTranspose Based on this, we can define a simplistic interpolation procedure (Figure 1.3), which determines the measure `?am_out` of a data element `?et_in` simply by “transposing” the measure of the nearest element in a given layer `?l_in`. More precisely, the query states that `?et_in` needs to have a measure `?am_out` that is `owl:sameAs` the measure of the data element `?e` in `?l_in` that is nearest to `?et_in`.

```

CONSTRUCT{
  ?nt wf:fstInput ?l_in;
  wf:sndInput ?et_in;
  wf:output ?am_out;
  a gis:NearTranspose.
}WHERE{
  ?l_in a gis:Layer.
  ?et_in ada:hasMeasure ?am_out.
  FILTER NOT EXISTS{
    ?l_in ada:hasElement ?e.
    ?e ada:hasMeasure ?att.
    ?n wf:output ?e;
    wf:fstInput ?et_in;
    wf:sndInput ?l_in;
    a gis:Nearest.
  }
  FILTER NOT EXISTS{ ?att owl:sameAs ?am_out. }
}}

```

Listing 3: NearTranspose

This one and other interpolation techniques (such Block Kriging) on the data item level are subclass of `gis:Interpolate` in the *gis* ontology. For the sake of brevity, the rest of the tools are described in Appendix A.

5. Computing query containment on a SPARQL subset

In this section, we suggest how query containment can be computed for a subset of the SPARQL query language that we consider particularly relevant for describing analytic ques-

tions in general, and GIS tools in particular. Our assumption is based on the observation that the diversity of analytic questions discussed in Section 2 and translated in Section 4.3 can be entirely expressed in this subset. In addition, we believe that GIS analyses tend to require questions that are structurally similar. We start with defining the subset in terms of a formal pattern, then specify query containment for this pattern, and finally propose an algorithmic solution.

5.1. $SPARQL_{CONSTRAINT}$

We call the subset of SPARQL that we propose here $SPARQL_{CONSTRAINT}$. This language allows expressing basic constraints as requirements on analysis results, in the form of conjunctions, negations and existential rules (Mugnier and Thomazo, 2014). Starting from the basics of the SPARQL syntax,²³ it can be defined as follows.

We denote a *triple pattern*, a triple of subject, predicate, and object in RDF, where any can be substituted by a variable,²⁴ with TP . A *basic graph pattern* is a conjunction of triple patterns:

Definition 2. *Basic graph pattern (BGP):*

$$TP_1 \wedge \dots \wedge TP_n$$

We introduce now special kinds of patterns for $SPARQL_{CONSTRAINT}$ in terms of basic graph patterns. The first one of these patterns is a *negated pattern*, which is simply a basic graph pattern with an (implicit) negation sign written in front:

Definition 3. *Negated graph pattern (NGP):*

$$\neg BGP$$

Note that in case the basic pattern is just a triple pattern, this simply becomes a negated atomic statement ($\neg TP$). In case of a more complex conjunction, the negation is equivalent to a disjunction of negated atomic statements ($\neg TP_1 \vee \dots \vee \neg TP_n$), meaning either one of the involved TP s must not be satisfied. A *negation set (NS)* is a conjunction of such negated graph patterns, asserting that each enclosed BGP must not be satisfied. If the set of negated patterns is empty, the NS is automatically satisfied (\top):

Definition 4. *Negation set (NS):*

$$NGP_1 \wedge \dots \wedge NGP_m \mid \top$$

Lastly, we also introduce a *rule pattern*. This is a tuple of basic graph patterns, where the first one acts as the body of the rule (*body pattern*), and the second one as the head of the rule (*head pattern*), and together they form an existential rule, where all variables in the body pattern are (implicitly) universally quantified, and all variables of the head pattern which do not appear in the body pattern are (implicitly) existentially quantified:

Definition 5. *Rule pattern (RP):*

$$\forall v_1, \dots, v_k. BGP_{body} \Rightarrow \exists v_l, \dots, v_z. BGP_{head}$$

A *rule set* is a conjunction of such rule patterns, stating that all rules must be satisfied by the query result:

Definition 6. *Rule set (RS):*

$$RP_1 \wedge \dots \wedge RP_o \mid \top$$

We now can define a $SPARQL_{CONSTRAINT}$ pattern as a conjunction of a basic graph pattern,

a negation set and a rule set, where the latter two might stay empty:

Definition 7. *Constrained graph pattern (CGP):*

$$BGP \wedge NS \wedge RS$$

A $\text{SPARQL}_{\text{CONSTRAINT}}$ CONSTRUCT query is a tuple of a *CGP* and a CONSTRUCT template which contains another BGP. In Appendix B, we show that a *CGP* corresponds to a certain subset of SPARQL.

5.2. Defining query containment for $\text{SPARQL}_{\text{CONSTRAINT}}$

In this subsection, we specify the precise conditions under which it is admissible to say that a given $\text{SPARQL}_{\text{CONSTRAINT}}$ query is a subquery of another. We start with introducing some common terminology.

A *solution mapping* for a query pattern is a function mapping the variables in this pattern into terms (RDF-T) such that the assertions of the pattern are preserved. A solution mapping binds the variables in the pattern to constants provided by data, and so matches the pattern with a data set. We call the existence of a solution mapping (in correspondence with logic terminology) a *model* of the pattern. So when we say a pattern has a model, it means there is a solution mapping, a non-empty query result. Note that a given pattern can have many different models. Furthermore, if a pattern does not have model, it must either be empty or contain a contradiction.

In order to determine whether one $\text{SPARQL}_{\text{CONSTRAINT}}$ query is contained in another one, we have to find out whether it is the case that *any model* (any query result) of the first query is also a model of the second query. We define query containment *in terms of homomorphic mappings* between query patterns which establish that they share all possible models. Homomorphic mappings are established separately for the different kinds of graph patterns that constitute a constrained graph pattern (CGP). We start with a BGP (compare Figure 4a):

Theorem 1. *A BGP_{con1} is contained in a BGP_{con2} , written $BGP_{con1} \subseteq BGP_{con2}$, iff there is a homomorphic mapping of all statements from BGP_{con2} into BGP_{con1} , such that all RDF terms are mapped into themselves (identity) and all variables are substituted either by RDF terms or by variables from V .*

Proof. Suppose there is such a homomorphic mapping m from BGP_{con2} into BGP_{con1} . By contradiction, assume that $\neg BGP_{con1} \subseteq BGP_{con2}$, i.e., there is a solution mapping μ for BGP_{con1} into a given set of RDF terms, but not for BGP_{con2} . Then consider the function $\mu_2 = m \circ \mu$. Since m is homomorphic, μ must be a solution for $m(BGP_{con2})$, and thus μ_2 must be a solution for BGP_{con2} , which contradicts our assumption. \square

To solve containment, it furthermore makes sense to take advantage of concept hierarchies in RDF, and thus of the reasoning capacities of Semantic Web ontologies. To add subsumption reasoning to a BGP mapping, simply expand BGP_{con1} with all inferable triples (using some ontology) before establishing the homomorphic mapping. Next, we define query containment for rule patterns and negated graph patterns.

Definition 8. *Sub-rule:*

A rule pattern RP_1 is a sub-rule of RP_2 , written $RP_1 \subseteq RP_2$, iff $BGP_{body2} \subseteq BGP_{body1}$ (the body of the super-rule is contained in the body of the sub-rule) and $BGP_{head1} \subseteq BGP_{head2}$ (the head of the sub-rule is contained in the head of the super-rule).

Note that the containment hierarchy is inversed for the rule's body (compare Fig. 4c): the

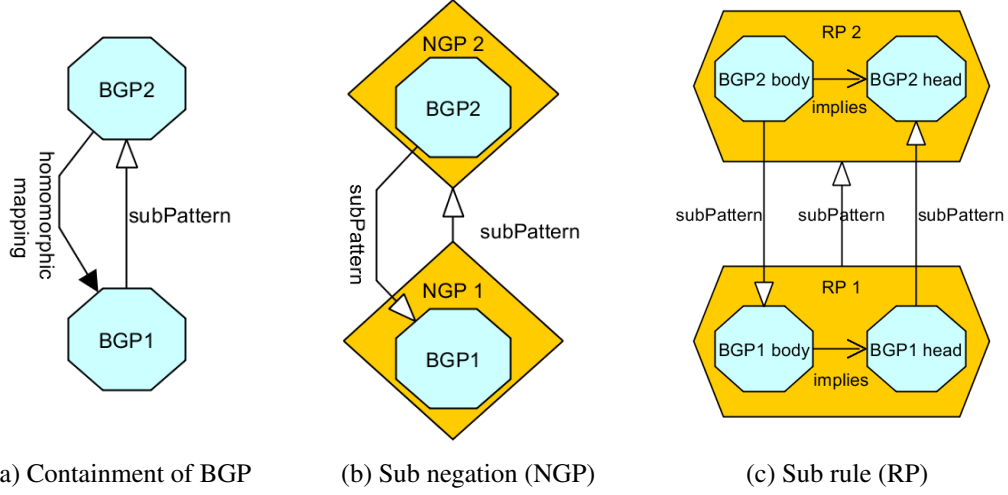


Figure 4.: Definitions of sub-query patterns for query containment of SPARQL constraint

condition of the sub-rule must contain the condition of the super-rule, in order to make sure that the sub-rule is always applicable whenever the super-rule is applicable to a data set, and if the sub-rule is not applicable, so is the super-rule. Since the head of the super-rule contains the head of the sub-rule, any introduction of new triples by the latter will automatically satisfy the super-rule's head.

Theorem 2. *If $RP_1 \subseteq RP_2$, then any model of RP_1 is also a model of RP_2 .*

Proof. By assumption, the sub-rule is satisfied by a model. Now suppose the super-rule's head is not applicable in this model. Then the super-rule is satisfied by definition. Otherwise, suppose it is applicable. Then by the definition of $RP_1 \subseteq RP_2$, the sub-rule's body must also be applicable, and since the sub-rule is satisfied as whole (by assumption), its head must be satisfied by the model. Again by the definition of $RP_1 \subseteq RP_2$, the head of RP_2 must be satisfied, too, and thus is RP_2 . \square

Definition 9. *Sub-negation:*

A negated graph pattern NGP_1 is a sub-negation of NGP_2 , written $NGP_1 \subseteq NGP_2$, iff $BGP_2 \subseteq BGP_1$ (the basic graph pattern of the super-negation is contained in the basic graph pattern of the sub-negation).

Note that in contrast to BGP containment, a negated graph pattern (NGP) with more triples than another NGP is always *more general*, not more specific. Note that therefore the containment relation is inversed with respect to the enclosed BGPs (see Fig. 4b). This is due to the fact that $\neg(A \wedge B)$ is $(\neg A \vee \neg B)$, which is a generalization of $(\neg A)$, not a specialization.

Theorem 3. *If $NGP_1 \subseteq NGP_2$, then any model of NGP_1 is also a model of NGP_2 .*

Proof. Suppose there is a model for NGP_1 . Then by definition, the corresponding BGP_1 is not satisfied by this model. By the definition of sub-negation, $BGP_2 \subseteq BGP_1$, and thus BGP_2 is not satisfied, too, which precisely satisfies NGP_2 . \square

Now we are ready to establish query containment for CGP:

Definition 10. *CGP query containment:*

A constrained graph pattern CGP_1 is contained in another pattern CGP_2 , written $CGP_1 \subseteq CGP_2$, if²⁵:

0) There is a mapping of all variables of CGP_2 into terms/variables of CGP_1 , and under this mapping,

1) $BGP_1 \subseteq BGP_2$

2) For each rule pattern RP_2^i in RS_2 , there is a rule pattern RP_1^j in RS_1 with $RP_1^j \subseteq RP_2^i$ (it contains the subrule pattern)

3) For each negated graph pattern NGP_2^i in NS_2 , there is a negated graph pattern NGP_1^j in NS_1 with $NGP_1^j \subseteq NGP_2^i$ (it contains the negated graph pattern)

In this definition, we first require a single mapping of variables into terms or variables for all subpatterns of a query. This makes sure we only consider models of the entire CGP. Otherwise, it would be possible to map subpatterns separately and inconsistently. We will see below that this makes an algorithmic solution less obvious. Note also that there may well be further rules/negations in CGP_1 that are not matched by any rule/negation in CGP_2 . Similarly, there may be triple patterns in BGP_1 that do not match any triple pattern in BGP_2 (since the homomorphic mapping needs not be surjective). All this simply means that CGP_1 can be more constrained than CGP_2 .

Theorem 4. If $CGP_1 \subseteq CGP_2$, then any model of CGP_1 is also a model of CGP_2 .

Proof. If CGP_2 consists only of a BGP, then by assumption, $BGP_1 \subseteq BGP_2$, and by Theorem 1, the model of BGP_1 is a model of BGP_2 and the empty rule set and negation sets are satisfied by assumption. If there is a rule in CGP_2 , it contains a sub-rule in CGP_1 by definition. Then by Theorem 2, the model of that sub-rule is also a model of the super-rule. In analogy, by Theorem 3, each model of a negated pattern NGP_1 contained by some NGP_2 is also a model of NGP_2 . Since all this is true under a common mapping of variables from CGP_2 into CGP_1 , every model of CGP_1 is also a model of CGP_2 . \square

5.3. Computing query containment for $SPARQL_{CONSTRAINT}$

Given the ideas outlined above, how can we decide whether a query is contained in another? For this purpose, we make use of the idea that the mappings defined above in Section 5.2 can be computed in terms of queries over queries. The principle idea of all the definitions in the last section is establishing homomorphic mappings between basic graph patterns which form various parts of a query. So whenever we find a way to compute such a mapping, we can design a procedure that divides a $SPARQL_{CONSTRAINT}$ query into its constituent parts and maps them to the respective parts of other queries. A mapping of a BGP, as used in Theorem 1, can in turn be established by a *query fired over a basic graph pattern*. That is, we suggest to use a SPARQL engine as a way to compute query containment.

However, there are three challenges to realize such an approach:

- First, it is necessary to turn a CGP query into an RDF graph against which we can fire BGP queries from other CGPs in order to establish the mapping. Thus we need a procedure to substitute variables in a BGP by RDF graph nodes and properties.
- Since mappings need to be established into both directions, from patterns of CGP_2 into patterns of CGP_1 and vice versa, queries must be fired separately and into both directions (see Fig. 4).
- Finally, it is not enough to establish mappings in this way for each CGP pattern part separately, since by Def. 10, variables must be mapped for the entire CGP pattern to obtain a global solution. However, we cannot fire entire CGP queries against each other.

To address the first challenge, we simply substitute variables by fake URIs, i.e., web addresses made of the variable names. Another option would be to substitute them with blank nodes. However, the former approach has the advantage that the variable is still identifiable across patterns and local contexts,²⁶ a necessary condition, as we will see below. To address the second challenge, we simply implement a mapping procedure which can be reversed. The last challenge is a bit more intricate, however, since it raises the complexity of the problem. Suppose a *CGP* with pattern parts $A \wedge B \wedge \dots$. If A is mapped using a query, then the solution of pattern B depends on the mapping of A , whenever A and B have variables in common. Our solution to this is as follows:

- (1) We map A using a query, storing variable bindings of the super-pattern into the sub-pattern.
- (2) We then iterate over these variable bindings, to substitute all the variables occurring in pattern B with the bindings of A .
- (3) We then map the 'concretized' pattern B using a query as in 1, and so forth for all parts in *CGP*.

When this procedure has successfully mapped each *CGP* part, we can be sure to have found a solution to the containment problem. Thus the procedure is correct. Note, however, that because Theorem 4 is only into one direction, this procedure is not complete. Algorithms 1, 2, 3, and 4 in Appendix C implement this approach.

6. Requesting GIS tools using questions

We implemented and tested our approach using the Python library *RDFlib*,²⁷ which was used to parse the SPARQL syntax in terms of SPARQL algebra,²⁸ as well as in order to query over *CGP* patterns of a statement. We furthermore used *RDFClosure* for RDFS reasoning on the level of BGP matching.²⁹ The code and data examples are available online.³⁰

Tools were described in terms of SPARQL constraint statements as suggested in Sect. 4.3, and requests were described on a higher level of abstraction, following the considerations in Sect. 4.2. The following requests were used to test the approach over these tool queries:

R1 What methods are available for interpolating all attribute measures of a target dataset from a given source data set? (Listing 4)

```

CONSTRUCT{
  ?method wf:output ?target_layer_in;
    wf:input ?layer_in.
}
WHERE{
  ?layer_in a ada:DataSet.
  ?target_layer_in a ada:DataSet.
  FILTER NOT EXISTS{
    ?target_layer_in ada:hasElement ?target_element.
    ?target_element ada:hasMeasure ?target_measure.
    FILTER NOT EXISTS{
      ?innermethod wf:input ?layer_in;
        wf:output ?target_measure;
        a gis:Interpolate.
    }
  }
}

```

Listing 4: R1: Request for interpolation tools that can handle whole data layers

For example, one may search for a method to interpolate measures of unemployment rates in election districts from a dataset of unemployment rates in administrative regions without

request	tool
requests/r1.rq	tools/defArealInterpolation.rq
requests/r2.rq	tools/defSnap.rq
requests/r3.rq	tools/defRasterResampling.rq
requests/r3.rq	tools/defViewshed.rq
requests/r3.rq	tools/defVRConversion.rq

Table 3.: Results of question-based tool requests R1 to R3

knowing exactly about the format of these datasets. Note that the head of the rule requests some “inner” interpolation operation in order to estimate these measures without specifying it. The result of matching this request over all tools can be seen in Table 3.

Areal interpolation (based on Block Kriging) is an adequate method to this end. However, other feature interpolation methods are possible.

R2 Which methods are available for enforcing some topological constraint on two geometries? (Listing 5)

```

CONSTRUCT{
  ?method wf:output ?geometry_out;
  wf:input ?geometry_in.
}WHERE{
  ?geometry_out a geo:Geometry.
  ?geometry_in a geo:Geometry.
  FILTER NOT EXISTS{
    FILTER NOT EXISTS{
      ?geometry_in gis:spatialTopRelation ?geometry_out.
    }
  }
}

```

Listing 5: R2: request for tools to enforce topological constraints

For example, one may search for a method to make sure that segments of a road network are properly connected at their boundary points in order to form a network. In this request, we search for editing methods that can be used to make sure geometries conform to a topological rule. This rule may have some arbitrary condition in the body, and so we leave the body of the rule empty. However, in the head, we request a statement about some topological relation between these geometries. We use a super-property for topological relations in GeoSPARQL to connect the two geometries in the head of the rule. The fact that one of the geometries is output shows that this is in fact a geometry editing method. The result of matching this request over all tools is in Table 3. It turns out that snapping is an adequate method to this end. Snapping assures that geometries touch each other under a distance condition. Note that the query would also find other tools with different topology rule conditions, such as object types.

R3 We search for methods that generate measures of a new raster based on some other layer (of whatever type). (Listing 6)

```

CONSTRUCT{
  ?method wf:input ?layer;
  wf:output ?raster_layer.
}WHERE{
  ?layer a ada:DataSet.
  ?raster_layer a gis:Raster.
  FILTER NOT EXISTS{

```



```

?cell ada:elementOf ?raster_layer;
      ada:hasMeasure ?cell_measure.
FILTER NOT EXISTS{
  ?innermethod wf:input ?layer;
               wf:output ?cell_measure.
}
}
}

```

Listing 6: Request for tools generating a raster layer from some other layer

For example, we may be interested in methods that derive a raster from, say, a set of maps of unknown format such as built environment, landuse and vegetation. The goal is to generate a spatially aligned raster with fixed extent and cell size from each data source, in order to later combine them into a cost surface for environmental analysis. It turns out (Table 3) that several tools correspond to this question, which might not be associated with the request when looking at them superficially. Viewshed analysis, raster conversion and raster resampling are normally used in very different GIS contexts. Yet, they all share the basic feature that they allow users to generate raster measures from some layer using some operation on the level of each individual raster cell. Thus they are meaningful candidates to accomplish the task.

7. Conclusion and outlook

In this article, we devised a semantic framework for the description of GIS tools in terms of the questions they answer. Our framework allows for the formulation of geospatial questions and the description of the high-level purpose of tools, regardless of the technology and implementation by focusing on the underlying questions. For this purpose, we defined a subset of the Semantic Web query language (*SPARQL_{CONSTRAINT}*) that captures conjunctions, negations and existential rules. These are particularly useful to formulate geospatial questions in terms of constraints on layer data elements, using known concepts of geometry or core concepts (Kuhn, 2012). We used CONSTRUCT queries to distinguish the question (in the WHERE clause) from the requested method that answers this question (the CONSTRUCT clause).

Our approach performs query containment resolution in this language to identify tools that answer user questions. We defined sufficient conditions for query containment and developed a correct, but non-complete algorithm that uses the SPARQL query engine to perform corresponding matchings. Given a knowledge base of tool descriptions and a formalized question, the algorithm identifies graph sub-patterns for each tool, translates them into RDF, and executes SPARQL queries over them in order to find matches.

To illustrate and test the approach, we described eight well-known GIS tools in terms of the questions they answer using *SPARQL_{CONSTRAINT}*. These annotations were tested against a set of user questions, showing that relevant tools are correctly retrieved. Questions were grounded in GIS practice from diverse applications. Thanks to its generic nature, the approach is extensible to many other tools and domains, such as data science, statistical analysis, engineering, architecture, and planning.

To make our framework fit for question-based retrieval and analysis, several areas of future work are worth pursuing. First, *SPARQL_{CONSTRAINT}* and our proposed ontology needs to be consolidated with respect to geospatial question formulation and tool descriptions. Is its expressiveness sufficient for other kinds of geospatial questions? More tools need to be documented using our framework in order to test the system with information retrieval measures. This gives us also a way to incrementally refine the interrogative spatial concepts of the ontology needed to bridge software specifics. A related future task is to add tool constraints on input data to a query, expressing considerations of meaningfulness (Scheider and Tomko,

2016).

Second, it is an open question how we could help ordinary GIS users and developers formulate questions and describe tools. In our framework, users still need to perform a manual abstraction step from a domain question to a tool request. Following the logic of query matching, a request needs to abstract from content themes and parameter values in order to subsume any tool description that is devoid of these specifics. Several approaches can be adopted to support and automate this step. For one, tool descriptions and questions need to be modularized, as done here by defining intermediate, inner methods and reusing them in other descriptions. This may result in a library of re-usable questions that are implementation-independent, as part of a linked method repository where tools can be registered with their corresponding questions (Scheider and Ballatore, 2018). Also, to increase the usability of our approach, controlled natural languages (Schwitter, 2010; Mazzeo and Zaniolo, 2016) and interactive SPARQL interfaces (Ngonga Ngomo et al., 2013) could be used to translate questions into queries, and autocompletion helps reuse existing interrogative concepts. Furthermore, we suggest to consider bottom-up approaches, such as query by example and case-based reasoning, in which a corpus of known questions is used to support the formulation of new ones and to automate the necessary abstraction to tool requests. Along the same lines, Web science can also help identify real usage patterns of tools and resources (Ballatore et al., 2018).

Third, the algorithm for query containment needs to be developed further to tackle issues like completeness, scalability, and performance. We currently use a brute-force approach to search over tools, which could be improved by reducing the search space of tools in question. To tackle completeness, we would need rule-based inference to derive queries from another query by the application of rules. Since this considerably increases the complexity of the algorithm, it should be carefully assessed whether practical applications really benefit from it (Hitzler and van Harmelen, 2010).

Finally, the integration of question-based analysis with linked GIS workflows remains an open problem (Scheider and Ballatore, 2018). How can we derive questions for entire workflows from questions over tools? Can we perform workflow composition and design using questions (Lamprecht, 2013) to solve indirect question answering? In our view, such efforts at question-based analytics have the potential to enable a more usable, inter-operable technological landscape for a more spatially-integrated data science.

Acknowledgements

We would like to thank Wim Feringa from ITC for the graphical design of Figure 1.

Notes

¹<http://blog.revolutionanalytics.com/2015/06/fishing-for-packages-in-cran.html>

²<http://desktop.arcgis.com>

³The latter follows a Zipf distribution, that is there are only few, simple most frequent queries, see Lin (2002).

⁴<https://www.w3.org/TR/sparql11-query/>

⁵<https://github.com/simonscheider/QuestionBasedAnalysis>

⁶<https://nyti.ms/2kc45DB>

⁷<http://desktop.arcgis.com/de/arcmap/10.3/tools/spatial-analyst-toolbox/raster-calculator.htm>

⁸For example, Ofoghi et al. (2008) suggested to use Fillmore's frames to match questions and answers.

⁹<https://en.wikipedia.org/wiki/SPARQL>

¹⁰See for example the Web Service Modeling Ontology (WSMO): <https://www.w3.org/Submission/WSMO/>

¹¹<http://desktop.arcgis.com/en/arcmap/10.3/map/working-with-layers/a-quick-tour-of-displaying-layers.htm>

¹²<https://www.w3.org/TR/sparql11-overview/>
¹³Web Ontology Language, <https://www.w3.org/TR/owl2-profiles/>
¹⁴We will mention example implementations from ArcGIS (<https://www.arcgis.com>) and ILWIS (<https://www.itc.nl/ilwis>).
¹⁵*ada*: <http://geographicknowledge.de/vocab/AnalysisData.rdf>
¹⁶*geo*: <http://www.opengis.net/ont/geosparql>
¹⁷*geof*: <http://www.opengis.net/def/function/geosparql/>
¹⁸We use the MathML *m*:<http://www.w3.org/TR/MathML/> “less than or equal” property (*m:leq*) to denote the filter function \leq .
¹⁹<https://github.com/simonscheider/QuestionBasedAnalysis>
²⁰*wf*: <http://geographicknowledge.de/vocab/Workflow.rdf>
²¹*gis*: <http://geographicknowledge.de/vocab/GISConcepts.rdf>
²²Note that this requires users to abstract from domain questions in order to formulate requests, see Section 6.
²³<https://www.w3.org/TR/rdf-sparql-query/>
²⁴see <https://www.w3.org/2001/sw/DataAccess/rq23/#BasicGraphPattern>
²⁵Note that we establish this only forward, not backward. The latter would require taking into account that a CGP query can be inferred from another using the application of rules. For example, from $\text{CGP}_a: \text{TP}_1 \wedge (\text{TP}_1 \Rightarrow \text{TP}_2)$, it follows that $\text{CGP}_b: \text{TP}_1 \wedge \text{TP}_2$ is always satisfied, and thus $\text{CGP}_b \subseteq \text{CGP}_a$.
²⁶Blank nodes loose their identity across local scopes.
²⁷<https://github.com/RDFLib/rdfLib>
²⁸<https://www.w3.org/2001/sw/DataAccess/rq23/rq24-algebra.html>
²⁹<https://github.com/RDFLib/OWL-RL>
³⁰<https://github.com/simonscheider/QuestionBasedAnalysis>

The supplemental material for this article is available at XXXXX.

References

- Albrecht, J. (1998). Universal analytical GIS operations: A task-oriented systematization of data structure-independent GIS functionality. In H. Onsrud and M. Craglia (Eds.), *Geographic information research: Transatlantic perspectives*, pp. 577–591. Abingdon, UK: Taylor & Francis.
- Allan, J., B. Croft, A. Moffat, and M. Sanderson (2012). Frontiers, challenges, and opportunities for information retrieval - Report from SWIRL 2012. *ACM SIGIR Forum* 46(1), 1–32.
- Ballatore, A., S. Scheider, and R. Lemmens (2018). Patterns of consumption and connectedness in GIS Web sources. In A. Mansourian, P. Pilesjö, L. Harrie, and R. van Lammeren (Eds.), *Geospatial Technologies for All. Selected papers of the 21st AGILE Conference on Geographic Information Science*, pp. 1–19. Berlin: Springer. In press.
- Belhajjame, K., J. Zhao, D. Garijo, M. Gamble, K. Hettne, R. Palma, E. Mina, O. Corcho, J. M. Gmez-Prez, S. Bechhofer, G. Klyne, and C. Goble (2015). Using a suite of ontologies for preserving workflow-centric research objects. *Web Semantics: Science, Services and Agents on the World Wide Web* 32, 16–42.
- Bernard, L., S. Mäs, M. Müller, C. Henzen, and J. Brauner (2014). Scientific geodata infrastructures: Challenges, approaches and directions. *International Journal of Digital Earth* 7(7), 613–633.
- Brauner, J. (2015). *Formalizations for Geooperators – Geoprocessing in Spatial Data Infrastructures*. Ph. D. thesis. Technical University of Dresden, Germany.
- Calì, A., G. Gottlob, and T. Lukasiewicz (2012). A general Datalog-based framework for tractable query answering over ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web* 14, 57–83.
- Canbek, N. G. and M. E. Mutlu (2016). On the track of artificial intelligence: Learning with intelligent personal assistants. *Journal of Human Sciences* 13(1), 592–601.
- Ferré, S. (2014). SQUALL: The expressiveness of SPARQL 1.1 made available as a con-

- trolled natural language. *Data & Knowledge Engineering* 94, 163–188.
- Fitzner, D., J. Hoffmann, and E. Klien (2011). Functional description of geoprocessing services as conjunctive datalog queries. *GeoInformatica* 15(1), 191–221.
- Gao, S. and M. F. Goodchild (2013). Asking Spatial Questions to Identify GIS Functionality. In *Proceedings of the Fourth International Conference on Computing for Geospatial Research and Application (COM.Geo)*, pp. 106–110. IEEE. San Jose, CA.
- Hinsen, K. (2014). Computational science: Shifting the focus from tools to models. *F1000Research* 3(101). <https://f1000research.com/articles/3-101/v1>.
- Hitzler, P., M. Krötzsch, and S. Rudolph (2009). *Foundations of Semantic Web Technologies*. Boca Raton, FL: CRC Press.
- Hitzler, P. and F. van Harmelen (2010). A Reasonable Semantic Web. *Semantic Web* 1(2), 39–44.
- Hofer, B., S. Mäs, J. Brauner, and L. Bernard (2017). Towards a knowledge base to support geoprocessing workflow development. *International Journal of Geographical Information Science* 31(4), 694–716.
- Höffner, K., J. Lehmann, and R. Usbeck (2016). CubeQA: Question Answering on RDF Data Cubes. In P. Groth, E. Simperl, A. Gray, M. Sabou, M. Krötzsch, F. Lecue, F. Flöck, and Y. Gil (Eds.), *15th International Semantic Web Conference (ISWC 2016)*, pp. 325–340. Kobe, Japan.
- Janowicz, K. (2016). *Modeling Ontology Design Patterns with Domain Experts-A View From the Trenches*. AKA Verlag.
- Kuhn, W. (2012). Core concepts of spatial information for transdisciplinary research. *International Journal of Geographical Information Science* 26(12), 2267–2276.
- Kuhn, W. and A. Ballatore (2015). Designing a Language for Spatial Computing. In F. Bacao, M. Y. Santos, and M. Painho (Eds.), *AGILE Conference on Geographic Information Science 2015, Lecture Notes in Geoinformation and Cartography*, pp. 309–326. Berlin: Springer.
- Kwan, M.-P. (Ed.) (2016). *Geographies of Health, Disease and Well-Being: Recent Advances in Theory and Method*. London: Routledge.
- Lamprecht, A.-L. (2013). *User-Level Workflow Design: A Bioinformatics Perspective*, Volume 8311 of *Lecture Notes in Computer Science*. Berlin: Springer.
- Lemmens, R., A. Wytzisk, R. By, C. Granell, M. Gould, and P. van Oosterom (2006). Integrating semantic and syntactic descriptions to chain geographic services. *IEEE Internet Computing* 10(5), 42–52.
- Lemmens, R. L. (2006). *Semantic interoperability of distributed geo-services*. Ph. D. thesis, Delft University of Technology, Delft, Netherlands.
- Lin, J. (2002). The Web as a Resource for Question Answering: Perspectives and Challenges. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC-2002)*, pp. 1–8. Canary Islands, Spain.
- Ludäscher, B., K. Lin, S. Bowers, E. Jaeger-Frank, B. Brodaric, and C. Baru (2006). Managing scientific data: From data integration to scientific workflows. *Geological Society of America – Special Papers* 397, 109–129.
- Lutz, M. (2007). Ontology-Based descriptions for semantic discovery and composition of geoprocessing services. *GeoInformatica* 11(1), 1–36.
- Mazzeo, G. M. and C. Zaniolo (2016). Answering Controlled Natural Language Questions on RDF Knowledge Bases. In *Proceedings of the 19th International Conference on Extending Database Technology (EDBT)*, pp. 608–611. Bordeaux, France.
- Mugnier, M.-L. and M. Thomazo (2014). An introduction to ontology-based query answering with existential rules. In M. Koubarakis, G. Stamou, G. Stoilos, I. Horrocks, P. Kolaitis, G. Lausen, and G. Weikum (Eds.), *Reasoning on the Web in the Big Data Era: 10th Inter-*

- national Summer School 2014, Athens, Greece*, pp. 245–278. Berlin: Springer.
- Müller, M., L. Bernard, and D. Kadner (2013). Moving code – Sharing geoprocessing logic on the Web. *ISPRS Journal of Photogrammetry and Remote Sensing* 83, 193–203.
- Ngonga Ngomo, A.-C., L. Bühmann, C. Unger, J. Lehmann, and D. Gerber (2013). Sorry, I don't speak SPARQL: Translating SPARQL queries into natural language. In *Proceedings of the 22nd International Conference on the World Wide Web (WWW'13)*, pp. 977–988. Rio de Janeiro, Brazil.
- Ofoghi, B., J. Yearwood, and L. Ma (2008). The impact of semantic class identification and semantic role labeling on natural language answer extraction. In C. Macdonald, I. Ounis, V. Plachouras, I. Ruthven, and R. W. White (Eds.), *Advances in Information Retrieval: 30th European Conference on IR Research, ECIR 2008, Glasgow, UK*, pp. 430–437. Berlin: Springer.
- OGC (2015). OGC WPS 2.0 Interface Standard. OGC document 14-065. Technical report, Open Geospatial Consortium, Wayland, MA.
- Rey, S. J. (2009). Show me the code: Spatial analysis and open source. *Journal of Geographical Systems* 11(2), 191–207.
- Rico, M., C. Unger, and P. Cimiano (2015). Sorry, I only speak natural language: A pattern-based, data-driven and guided approach to mapping natural language queries to SPARQL. In *Proceedings of the 4th International Workshop on Intelligent Exploration of Semantic Data (IESD 2015) co-located with the 14th International Semantic Web Conference (ISWC 2015)*, pp. 1–10.
- Scheider, S. and A. Ballatore (2018). Semantic Typing of Linked Geoprocessing Workflows. *International Journal of Digital Earth* 11(1), 113–138.
- Scheider, S. and R. Lemmens (2017). Using SPARQL to describe GIS methods in terms of the questions they answer. In A. Bregt, T. Sarjakoski, R. van Lammeren, and F. Rip (Eds.), *Short papers, posters and poster abstracts of the 20th AGILE Conference on Geographic Information Science*, pp. 1–6. Wageningen, Netherlands.
- Scheider, S. and M. Tomko (2016). Knowing Whether Spatio-Temporal Analysis Procedures Are Applicable to Datasets. In *Proceedings of the 9th International Conference on Formal Ontology in Information Systems, FOIS 2016*, pp. 67–80. Annecy, France.
- Schwitter, R. (2010). Controlled natural languages for knowledge representation. In *COLING '10 Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, pp. 1113–1121. Association for Computational Linguistics. Beijing, China.
- Visser, U., H. Stuckenschmidt, G. Schuster, and T. Vogeles (2002). Ontologies for geographic information processing. *Computers & Geosciences* 28, 103–117.
- Zhao, P., T. Foerster, and P. Yue (2012). The geoprocessing web. *Computers & Geosciences* 47, 3–12.

Appendix A

Areal interpolation The areal interpolation query (Listing 7) makes use of some interpolation on data item level, and scales it up to the level of a whole (regional) layer. This operation takes as (first) input a layer `?l_in` based on which the interpolation is done. The second input is the target layer `?ltgt_in` whose measures need to be interpolated from the first layer. The query states that all measures of the target layer need to be the result of some interpolation (in this case, a *Block Kriging*) from the first input layer (Figure 1.4). A variation of this operation could use *NearTranspose* defined in Listing 3.

```
CONSTRUCT{
  ?ai wf:fstInput ?l_in;
    wf:sndInput ?ltgt_in;
    wf:output ?ltgt_in;
    a gis:ArealInterpolation.
}WHERE{
  ?l_in a gis:RegionDataSet.
  ?ltgt_in a gis:RegionDataSet.
  FILTER NOT EXISTS{
    ?ltgt_in ada:hasElement ?ftgt_out.
    ?ftgt_out ada:hasMeasure ?msr_out.
    FILTER NOT EXISTS{
      ?i wf:fstInput ?l_in;
        wf:sndInput ?ftgt_out;
        wf:output ?msr_out;
        a gis:BlockKriging.
    }
  }
}
```

Listing 7: Areal interpolation

Snap Snapping a geometry `?snappedgeom_out` to another one (`?geometry_in`) (Figure 1.5) leaves it in a state where all of its boundary points are either outside of the given snapping distance (`?snapdist_in`) to `?geometry_in`, or part of the boundary of `?geometry_in`. The snapping distance is a (metric) parameter of the method (see Listing 8). It is important to stress that this query does not provide an algorithmic implementation of snapping, but it only describes the required result. Note that we use the GeoSPARQL property `geo:sfTouches` to say that geometric boundaries share points (e.g. lines share endpoints, or regions share border points).

```
CONSTRUCT{
  ?sn wf:fstInput ?geometry_in;
    wf:output ?snappedgeom_out;
    gis:param ?snapdist_in;
    a gis:Snap.
}WHERE{
  ?snappedgeom_out a geo:Geometry.
  ?geometry_in a geo:Geometry.
  FILTER NOT EXISTS{
    ?d a geof:distance;
      wf:fstInput ?geometry_in;
      wf:sndInput ?snappedgeom_out;
      gis:param xsd:meter;
      wf:output ?dist.
    ?dist m:leq ?snapdist_in.
    FILTER NOT EXISTS{
      ?geometry_in geo:sfTouches ?snappedgeom_out
    }
  }
}
```

```
}}}
```

Listing 8: Snap

Viewshed Analysis A viewshed analysis calculates which cells of a raster are visible from some observer point (Figure 1.6). That is, all cells in the output raster with value 1 must be visible. In Listing 9, the output is a raster (?visraster_out), in which cells have value 1 if the cell is visible, or 0 otherwise. A viewpoint is indicated with variable ?viewpoint_in. Visibility is captured by a complex (3-ary) relation `gis:Visible` between a viewpoint, a height raster ?raster_in and a visible cell location ?cellgeom.

```
CONSTRUCT {
  ?vs wf:fstInput ?raster_in;
  wf:sndInput ?viewpoint_in;
  wf:output ?visraster_out;
  a gis:Viewshed.
}WHERE{
  ?viewpoint_in a gis:Point.
  ?raster_in a gis:Raster.
  FILTER NOT EXISTS{
    ?cell ada:elementOf ?visraster_out;
    ada:hasMeasure "1";
    ada:hasSupport ?cellgeom.
    FILTER NOT EXISTS{
      ?vr a gis:Visible;
      gis:from ?viewpoint_in;
      gis:height ?raster_in;
      gis:visible ?cellgeom.
    }
  }
}}
```

Listing 9: Viewshed analysis

Vector Raster Conversion This method converts a vector layer into a raster layer, following some spatial interpolation strategy given as parameter (Figure 1.7). One possibility would be `NearTranspose`, that is, assigning simply the measure of the nearest vector element, but there are also other possibilities (e.g. midpoint based, or percentage of overlap). The filter rule in Listing 10 states that for all cells within ?raster_out, their attribute is interpolated from the vector layer, leaving open how the interpolation could actually be done. Note that these SPARQL queries need not contain the implementation details of the operation, which can be performed with several alternative approaches.

```
CONSTRUCT{
  ?ftr wf:fstInput ?vector_in;
  gis:param ?intpl;
  wf:output ?raster_out;
  a gis:FeatureToRaster.
}WHERE{
  ?vector_in a gis:Vector.
  ?raster_out a gis:Raster.
  ?intpl a gis:Interpolate.
  FILTER NOT EXISTS{
    ?cell ada:elementOf ?raster_out;
    ada:hasMeasure ?attribute.
    FILTER NOT EXISTS {
      ?intpl wf:fstInput ?vector_in;
      wf:sndInput ?cell;
    }
  }
}
```

```

wf:output ?attribute.
}}}

```

Listing 10: Vector Raster Conversion (midpoint)

Raster Resampling This operation (Figure 1.8) transforms a raster layer to a new (non-coinciding) raster layer, resampling the cell values using some interpolation technique. The option that is most often used is `NearTranspose`. In Listing 11, the input layer is a raster (`?l_in`), to be resampled into a target raster (`?l_out`). Note the logical similarity of the two last queries, even though they are performed on different data formats.

```

CONSTRUCT{
  ?rr wf:fstInput ?l_in;
  gis:param ?intpl;
  wf:output ?l_out;
  a gis:RasterResampling.
}WHERE{
  ?l_in a gis:Raster.
  ?l_out a gis:Raster.
  ?intpl a gis:Interpolate.
  FILTER NOT EXISTS{
    ?l_out ada:hasElement ?eo.
    ?eo ada:hasMeasure ?attribute.
    FILTER NOT EXISTS{
      ?intpl wf:fstInput ?l_in;
      wf:sndInput ?eo;
      wf:output ?attribute.
    }
  }
}

```

Listing 11: Raster Resampling

Appendix B

We will show that a CGP pattern logically corresponds to a particular kind of SPARQL FILTER query without any of the following additional SPARQL syntactic constructs: group patterns, OPTIONAL patterns, Minus, property paths, BIND, VALUES, subqueries, aggregates (GROUP BY statements), any other functions (including FILTER functions) and modifiers (ORDER BY, ...).

Theorem 5. *Any SPARQL graph pattern containing exactly one BGP as well as a (possibly empty) list of FILTER NOT EXISTS statements, where each statement possibly nests one other FILTER NOT EXISTS statement, can be translated into a CGP and vice versa.*

Proof. Forward: Turn the BGP into a conjunction. If there is no other pattern, this already corresponds to a CGP with empty NS and RS patterns. Otherwise, for each FILTER NOT EXISTS statement in the list, do the following: if the pattern inside the statement is not nested, turn it into a negated conjunction and add it to the conjunction NS. Otherwise, turn the basic graph pattern inside the outermost statement into a conjunction called "body", and the basic graph pattern inside the innermost FILTER NOT EXISTS statement into a conjunction called "head", form the tuple (body,head), and add it the conjunction RS. Then form a conjunction of BGP, NS and RS. Backward: Since the procedure defined above is 1 to 1, it can always be reversed. \square

Appendix C

The following algorithms were implemented in Python and are available on github³¹. Note that in Algorithms 2 and 3, an empty super-pattern (ns2) will result in a positive result (an empty result list), not *False*. Thus an empty negation set or rule set is satisfied by default. In Algorithm 4, we illustrated variable substitution for negation patterns, leaving away rule patterns for reasons of space and because they are treated in an equivalent manner. The function *getVariableMap* retrieves variable bindings from a result set of a query, and the function *substituteVars* performs variable substitution in the CGP pattern that was used to construct the BGP query (cgp2). Note that in the NGPs case, we need to invert (function *invertmap*) the variable binding, because we fire the query vice versa, from *cgp* to *cgp2*. Note also that we added a heuristic simplification to the procedure by storing only the first matching variable binding of the NGPs subpatterns. This speeds up the process without affecting the correctness of the program. While it may affect the recall behavior under certain circumstances, in our tests, the quality of our results was not affected at all, because the variability of matching NGPs and RPs is usually very restricted.

Algorithm 1: BGP, NGP and RP query containment

```

subBGP (bgp, bgp2)
    Result: Decide whether bgp is a subpattern of bgp2
    /* Empty pattern is subpattern of any pattern */
    if bgp == [] then
        return True;
    end
    /* Substitute variables by fake URIs ... */
    rdf = BGP2RDF(bgp) ;
    /* Turn into Select* query ... */
    ask = BGP2ASK(bgp2) ;
    return rdf.query(ask) ;
subNGP (ngp, ngp2)
    Result: Decide whether ngp is a subpattern of ngp2
    return subBGP(ngp2,ngp);
subRP (rp, rp2)
    Result: Decide whether rp is a subpattern of rp2
    body2 = rp2[0]; head2 = rp2[1];
    body1 = rp[0]; head1 = rp[1];
    return [subBGP(body2, body1),subBGP(head1, head2)];

```

Algorithm 2: NS containment

```

subNS (ns, ns2)
resultlist = [];
for pa in ns2 do
    res = False;
    for p in ns do
        result = subNGP(p, pa);
        if bool(result) then
            resultlist.append(result);
            res = True;
            break;
        end
    end
    if not res then
        | return False
    end
end
return resultlist;

```

Algorithm 3: RS containment

```

subRS (rs, rsp2)
resultlist = [];
for pa in rs2 do
    res = False;
    for p in rs do
        result = subRP(p, pa);
        if bool(result[0] and result[1]) then
            resultlist.append(result);
            res = True;
            break;
        end
    end
    if not res then
        | return False
    end
end
return resultlist;

```

Algorithm 4: CGP query containment

```

subCGP (cgp, cgp2)
Result: Decide whether cgp is a subpattern of cgp2
varmaps = [];
/* Check query containment for BGP */
bgpresult = subBGP(cgp.bgp,cgp2.bgp);
if bool(bgpresult) then
    /* BGP Match! Store variable bindings */
    varmaps = getVariableMap(cgp2,bgpresult)
else
    return False /* 'BGP does not Match!'
end
/* Check query containment for NGPs */
varmapk = [];
if varmaps != [] then
    /* First substitute variables with bindings */
    for varmap in varmaps do
        cgp2ngpsubs = [];
        for triples in cgp2.ngps do
            cgp2ngpsubs.append(substituteVars(triples, varmap));
        end
        ngpresults = subNS(cgp.ngps,cgp2ngpsubs);
        if ngpresults != False then
            /* NGPs Match! Store variable bindings */
            for ngpresult in ngpresults do
                | varmap.update(invertmap(getVariableMap(cgp,ngpresult)[0]))
            end
            varmapk.append(varmap);
        else
            return False /* NGPs do not Match!
        end
    end
else
    ngpresults = subNS(cgp.ngps,cgp2.ngps);
    if ngpresults != False then
        /* NGPs Match! Store variable bindings */
        for ngpresult in ngpresults do
            | varmapk.append(invertmap(getVariableMap(cgp,ngpresult)[0]))
        end
    else
        return False /* NGPs do not Match!
    end
end
/* contd' for RPs...
return True

```