Adding Completeness Information to Query Answers over Spatial Databases

Simon Razniewski
Free University of Bozen-Bolzano
Dominikanerplatz 3
39100 Bozen, Italy
razniewski@inf.unibz.it

Werner Nutt
Free University of Bozen-Bolzano
Dominikanerplatz 3
39100 Bozen, Italy
nutt@inf.unibz.it

ABSTRACT

Real-life spatial databases are inherently incomplete. This is in particular the case when data from different sources are combined. An extreme example are volunteered geographical information systems like OpenStreetMap.

When querying such databases the question arises how reliable are the retrieved answers. For instance, for positive queries, which ask for existing patterns of objects, further answers could show up if the data is completed. For queries with negation, it is furthermore possible that after data completion objects cease to satisfy a query.

On the OpenStreetMap wiki, contributors have started to record for some areas which object types have been mapped completely. Given a query, we show how such metainformation can be used to classify objects in the database as certain answers, which are certainly answers in reality, impossible answers, which in reality are definitely not answers, and possible answers, for which it is not known whether they are answers in reality. In addition, we compute the completeness area of a query, that is the maximal area for which it is certain that no further answer objects exist in reality.

All this additional information can be computed with standard operations on spatial data. Experiments suggest that the computation of such completeness information is feasible.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: [Database Applications – Spatial databases and GIS]

Keywords

Data Quality, Data Completeness, Metadata Management

1. INTRODUCTION

Storing and querying geographic information poses additional requirements on databases that motivated the development of dedicated architectures and algorithms for spatial data management. Recently, due to the increased availability of GPS devices, volunteered geographical information systems have quickly evolved, with OpenStreetMap (OSM) being the most prominent one. Ongoing open public data initiatives that allow to integrate government data also contribute. The level of detail of OpenStreetMap is generally significantly higher than that of commercial solutions such as Google Maps or Bing Maps, while its accuracy and completeness are comparable.

OpenStreetMap allows to collect information about the world in remarkable detail. This, together with the fact that the data is collected in a voluntary, possibly not systematic manner, brings up the question of the completeness of the OSM data. When using OSM, it is desirable also to get metadata about the completeness of the presented data, in order to properly understand its usefulness.

Assessing completeness by comparison with other data is only possible if a more reliable data source for comparison exists, which is generally not the case. Therefore, completeness can best be assessed by metadata about the completeness of the data that is produced in parallel to the base data, and that can be compiled and shown to users. When providing geographical data it is quite common to also provide metadata, e.g., using standards such as the FGDC metadata standard[1]. However, little is known about how this metadata can be used to annotate query answers with completeness information.

As an example, consider that a tourist is looking for hotels that are nearby a park. She can query the OSM database, but due to the open nature of information in OSM, she does not get any information about how good the query result is wrt. the reality. It could be both the case that hotels, that according to OSM do not have a park nearby, do have a park nearby in reality, and also that in reality there are further hotels with parks nearby, which are not mapped in OSM.

For queries that ask for the absence of features, such as hotels which are not near a factory, the situation is even worse. Due to the open nature of OSM, one cannot get any guarantees whether a hotel is nearby a factory or not.

To make conclusions about the completeness of query answers, one needs completeness metadata. Our contribution in this paper is as follows: We show that when information about completeness is present, two things can be done:

(i) It is possible not only to identify objects that certainly satisfy a query even when the query asks for the absence of other objects, but also to split objects present in the data into those that can possibly satisfy the query and those for which that is impossible.

[1]http://www.fgdc.gov/metadata/geospatial-metadata-standards

SIGSPATIAL '14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA

http://dx.doi.org/10.1145/2666310.2666395.

(ii) One can identify in which areas of the map the reality cannot contain any further answers.

We also show that metadata about completeness is already present to a limited extent for OSM, and discuss practical challenges regarding acquisition and usage of completeness metadata in the OSM project.

The structure of this paper is as follows: In Section 2, we present a sample scenario, in Section 3 we give background information about spatial databases, OpenStreetMap and geographical data completeness. In Section 4, we formalize spatial databases, queries, completeness statements and answer classes. In Section 5, we present results for reasoning, show experimental results in Section 6 and discuss practical aspects in Section 7.

The idea of annotating query answers with completeness information appeared first in [10]. This work however presented only a vague formalization of the problem, considered an intractable framework including arithmetic comparisons, and contained no decision procedures. Also, it did not discuss queries with negation.

2. MOTIVATING SCENARIO

OpenStreetMap is a popular volunteered geographical information system that allows access to its base data to anyone. To coordinate their efforts, the creators (usually called Mappers) of the data use a wiki to record the completeness of objects in different areas. We want to show that this information can also be interesting for users that query the data.

Example 1. As a particular use case, consider that a user Mary is planning vacations in Abingdon, UK[2]. Assume Mary is interested in finding a 4-star hotel that is near a public park. Using the Overpass API[3], she could formulate in XML the following query[4] and execute it online over the OSM database:

```
<query type="node">
  <has-kv k="leisure" v="park"/>
  <bbox-query {{bbox}}/>
</query>
<query type="node">
  <around radius="2000"/>
  <has-kv k="tourism" v="hotel"/>
  <has-kv k="stars" v="4"/>
   <bbox-query {{bbox}}/>
</query>
<print/>
```

Suppose now that the data stored about Abingdon looks like shown in Fig. 1, that is, there are three four-star hotels, the Moonshine Star, the British Rest and the Holiday Inn, and two parks, King's Garden and Central Park in the database.

In this example, Mary's query would return only the hotel Moonshine Star as answer (Fig. 2).

What can be said about the quality of this answer? Can Mary ignore the British Rest and the Holiday Inn hotels? Could there be possibly other hotels in Abingdon that would match her query?

[2]We chose Abingdon here because the OSM wiki contains generally much better information about small towns than about big towns.
[3]http://overpass-turbo.eu/
[4]This particular query does not return any answer for Abingdon, but e.g. for Berlin or Paris it does.

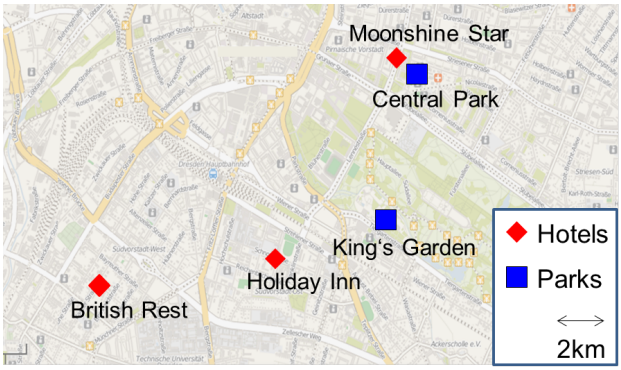

Figure 1: Sample database for Abingdon.

Figure 2: Result for Mary's query for hotels near a park.

Or, suppose Mary changes her mind and instead searches for hotels that are not near a factory. For this query, the OSM database would return all three hotels as answers. But does that mean that there is really no factory near these hotels? Or can it be the case that factories have not been mapped in Abingdon, and that there is one near each hotel?

Without further background information these questions cannot be answered. Mary therefore browses the wiki of OSM, where she finds a page[5] containing information about the completeness of data about Abingdon (Fig. 3). What do the symbols on this page mean? On another page[6], she finds an explanation of the symbols (Fig. 4).

What can Mary deduce from this information about the completeness of the result of her query? This will be discussed in the remainder of this paper.

Another use case where completeness is important could be emergency planning, where the planners are interested to find all schools that are are within a certain radius of a chemical industry complex. Schools may be completely mapped only in parts of the area of interest. Therefore, to assess in which areas the query answer is complete, they would need metadata telling in which areas all schools have been mapped. Also, areas where industry complexes have been mapped completely, and no industry complex exist would be of interest, because in these areas, irrespective of whether schools are complete or not, no school can be close to an industry complex.

[5]http://wiki.openstreetmap.org/wiki/Abingdon
[6]http://wiki.openstreetmap.org/wiki/Template:En:Map_status

Community	Slice #/ Description	Status	Remarks	Mapped/checked by
Abingdon	1. Central + Ock St. to R. Ock		Other tourist stuff besides the museum? Bridge Street detail has been missed: Footpaths by river near the bridge. User:greenius 16 November 2008	Mapped:User:Achadwick (based on earlier)
Abingdon	2. Caldecott, towards Culham			
Abingdon	3. Business park, cemeteries, Willow Brook devel, Albert Park		... Need to check for cycle lanes, etc along Marcham Road.	Mapped:User:Achadwick (added to existing) Checked:User:Duncanparkes (with a few additions)
Abingdon	4. Residential triangle, Longmead etc.		Pub is only restaurant? Footways that link stuff, stubbed in places.	Mapped:User:Achadwick (started)
Shippon	5. Whole village, minus the barracks		Mostly done here.	Mapped:User:Achadwick (based on others' work)

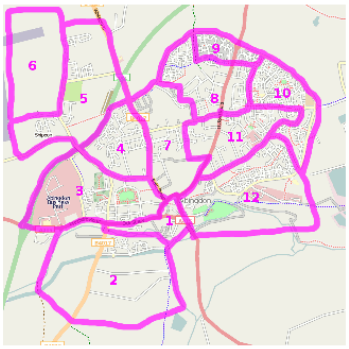

Figure 3: Extract from the OpenStreetMap wiki page for Abingdon. Source: http://wiki.openstreetmap.org/wiki/Abingdon

Colour and symbol legend

The status of each map region is indicated by a symbol, which describes the type of feature, and a colour, which indicates the completeness of that feature in a map region.

Meaning of symbols

- - Roads for car traffic are present. One way streets and pedestrian streets are present. This means that the map can be used for car navigation - Key: c
- - Street names are labelled. This means that the map can be used to find an address - Key: l
- - All housenumbers are present - Key: h
- - All cycleways, and field and forest roads suitable for bicycles are present - Key: b
- - All footways are present - Key: fo
- - All wheelchair access is present - Key: d
- - All public transports are present (including name of the bus stop/station names) - Key: tr
- - All public institutions are present (Churches, sports facilities, venues, schools, hospitals, city hall) - Key: p
- - All petrol stations are present - Key: fu
- - All restaurants and hotels are present - Key: r
- - All tourist attractions are present - Key: t
- - All natural resources are mapped (e.g Water, Lakes and Woodland) - Key: n

Meaning of colours

Background colour	Meaning	Use for navigation	To do	value
	The map needs checking, status unknown	Availability unknown	Please check	(None)
	The map contains no or little data	Not to be used	Please complete	0
	The map contains partial data	Limited usability	Please complete	1
	The map is largely complete (please describe missing data)	Use with restrictions	Please complete (missing data, streets etc.)	2
	The map is complete (in the opinion of a mapper)	Suitable for use	Please check and correct any errors	3
	The map is complete (verified by 2 mappers) ; please indicate Date when checked)	Suitable for use	Please update as needed	4
	This attribute does not exist in the mapped area (e.g. no petrol stations)	Suitable for use	Please update as needed	X

Figure 4: Legend for completeness statements as shown on the OpenStreetMap wiki page. Source: http://wiki.openstreetmap.org/wiki/Template:En:Map_status

3. BACKGROUND

In the following, we introduce spatial database systems, OpenStreetMap, and the problem of geographical data completeness.

3.1 Spatial Databases and OpenStreetMap

To facilitate storage and retrieval, geographic data is usually stored in *spatial databases*. According to [2], spatial databases have three distinctive features. First, they are database systems, thus classical relational/tree-shaped data can be stored in them and retrieved via standard database query languages. Second, they offer spatial data types, which are essential to describe spatial objects. Third, they efficiently support operations on spatial data types via spatial indexes and spatial joins.

OpenStreetMap (OSM) is a free, open, collaboratively edited map project. Its organization is similar to that of Wikipedia. Its aim is to create a map of the world. The map consists of objects (there called features) which have associated geometries, which are either points, polygons or groups of the former two, and each object has a primary type (category), such as highway, amenity or similar. Furthermore, each object can have an unrestricted set of key-value pairs. Though there are no formal constraints on the key-value pairs, there are agreed standards for each primary object type.[7]

There have been some assessments of the completeness of OSM based on comparison with other data sources, which showed that the road map completeness is generally good [4, 3, 7]. Assessment based on comparison is however a method that is very limited in general, as it relies on a data source that captures some aspects equally well as OSM. Especially since due to the open key-value scheme the level of detail of OSM is not limited, comparison is not possible for many aspects. Examples of the deep level of detail are the kind of trash that trash bins accept or the opening hours of shops or the kind of fuel used in public fire pits [8] (these attributes are all agreed as useful by the OSM community).

While the most common usage of OSM is as online map service, it also provides advanced querying capabilities, for instance via the Overpass API web interface. Also, the OSM data, which is natively in XML, can be downloaded, converted and loaded into classical SQL databases with geographical extensions.

3.2 Geographical Data Completeness

Geographical data quality is important, as for instance recent media coverage on Apple misguiding drivers into remote Australian desert areas shows.[9] There has long been work on geographical data quality, however it was mostly focusing on precision and accuracy [11]. Completeness poses the challenge that it may highly vary depending on the type of object. If metadata about completeness is present, it is attractive to visualize it on maps [12]. Completeness is especially a challenge when (1) databases are to capture continuously the current state (as opposed to a database that stores a map for a fixed date) because new objects can appear, (2) databases are built up incrementally and are accessible during build-up (as it is the case for OSM) and (3) the level of detail that can be stored in the database is high (as it is easier to be complete for all highways in a state than for all post boxes).

Work on analyzing the completeness of OpenStreetMap was done by Mooney et al. [7] Haklay and Ellul [4, 3] and Zielstra et al. [13]. The first work introduced general quality metrics for OSM, while the latter works analyzed the completeness of the road maps in England and in the US by comparing them with government data sources, finding that each data source was better than the other in some aspects, and worse in others.

To the best of our knowledge, regarding metadata-based completeness assessment of query answers over geographical data, no work has been done so far.

3.3 Incompleteness in Database Theory

In database theory, there has been extensive work on incompleteness. The core framework was established by Imielinski and Lipski [5], who introduced the terms of certain and possible answers. Note that we are using the terms differently here, as in the classical framework, certain answers are a subset of possible answers, whereas in our work, the two are disjoint.

Completeness statements and completeness reasoning were first introduced by Motro [8], who used statements about the completeness of query answers to infer the completeness of other queries, and Halevy [6], who used statements about the completeness of parts of a database to infer completeness of query answers. Later work by Razniewski and Nutt [9] provided decision procedures for the problem introduced by Halevy. The completeness statements that we use in this paper are an adaption of a simple case (condition-free) of the statements introduced by Halevy, although our conclusions are very different, because we consider the state of the database, whereas the work of Halevy was on the level of the schema only.

4. FRAMEWORK

In the following, we review the notion of spatial databases, introduce distance queries as object of study in this paper, and present a framework for completeness statements over spatial databases and for the annotation of query answers with completeness information.

4.1 Standard Definitions

While the data format of OSM allows to add arbitrary key-value pairs to objects, there exists a community consensus on the common attributes of different object categories.

Using these agreed attributes, the data can then be transformed into relational data, thus, in the following, we adopt a relational database view.

Spatial databases consist of sets of objects, which are formulated using a fixed vocabulary, the database schema. Each object has one *location* attribute. For simplicity, we assume that these locations are only points.

We assume a fixed set of object names Σ, where each object name R has a set of arbitrary attributes and one location attribute. Then, a *spatial database* is a finite set of facts over Σ that may contain null values. Null values correspond to key/value pairs that are not set for a given object.

Example 2. Represented in a spatial database D_{Abgd}, the information from Fig. 1 could looks as follows:

[7] http://wiki.openstreetmap.org/wiki/Map_features
[8] http://wiki.openstreetmap.org/wiki/Tag:amenity%3Dbbq
[9] http://www.dailymail.co.uk/sciencetech/article-2245773/Drivers-stranded-Aussie-desert-Apple-glitch-Australian-police-warn-Apple-maps-kill.html

Hotel			
name	stars	restaurant	location
Moonshine Star	4	yes	48.5527:9.6481
British Rest	4	yes	48.1220:9.5804
Holiday Inn	4	no	48.4176:9.3721

Park		
name	size	location
Central Park	med	48.2082:9.5771
King's Garden	small	48.4908:9.6148

In the following, we employ a Datalog-style [1] notation for queries.

A *simple query* over a spatial database is written as $Q(\bar{t}, l) :- R(\bar{t}, l)$, where R is an object type, the terms $\bar{t}$ are either constants or distinct variables and l is the location attribute of R.

Example 3. A simple query asking for hotels with 4 stars is written as follows:

$$Q_{4\text{stars}}(n, 4, r, l_{hotel}) :- hotel(n, 4, r, l_{hotel}).$$

Spatial query languages allow the use of spatial relations and functions such as distance, growing and shrinking.

Over spatial databases, it is especially interesting to retrieve objects for which there exists another object of a specific type within a certain proximity, or for which no object of a specific type exists within a certain proximity. To express such queries, we introduce the class of so-called distance queries, on which we will focus in the remainder of this paper:

Intuitively, a *distance query* asks for an object for which specific other objects exist within a certain radius. In this type of query, joins between atoms appear only between the location of the first object and the locations of the other objects. Formally, a positive distance query with $n+1$ literals is written as follows:

$$\begin{aligned} Q(\bar{t}_0, l_0) :- R_0(\bar{t}_0, l_0),\ & R_1(\bar{t}_1, l_1),\ \text{dist}(l_0, l_1) < d_1, \\ & R_2(\bar{t}_2, l_2),\ \text{dist}(l_0, l_2) < d_2, \\ & \dots \\ & R_n(\bar{t}_n, l_n),\ \text{dist}(l_0, l_n) < d_n \end{aligned} \tag{1}$$

where l_i is the geometry attribute of the object R_i, the $\bar{t}_i$ are tuples of constants and distinct variables, and the d_i are constants. We call $R_i(\bar{t}_i, l_i)$ the literal L_i. We will refer to the literal L_0 as the *core* of the query, and to the other literals as the *satellites* of the query.

Later, we will also discuss queries with negated atoms. Note that using the relations '$\neq$' and '$=$' together with dist does not make sense for a nearly continuous-valued attribute such as location, and that the condition 'dist $> d$' does not make practical sense, because in order to evaluate such a query, one would need to scan the objects in the whole world.

Example 4. Consider again Mary's query that asked for 4-star hotels with a park within two kilometres distance. In Datalog, this distance query would be written as follows:

$$\begin{aligned} Q_{\substack{\text{nice}\\ \text{Hotel}}}(n, 4, r, l_{hotel}) :- & \ hotel(n, 4, r, l_{hotel}),\ park(n', s', l_{park}), \\ & \text{dist}(l_{hotel}, l_{park}) < 2km. \end{aligned}$$

A query that additionally asks for pubs within one kilometre and a train station within one kilometre would be written as follows:

$$\begin{aligned} Q_{\substack{\text{nicer}\\ \text{Hotel}}}(n, 4, r, l_{hotel}) :- & \ hotel(n, 4, r, l_{hotel}), \\ & park(n', s', l_{park}),\ \text{dist}(l_{hotel}, l_{park}) < 2km \\ & pub(n'', l_{pub}),\ \text{dist}(l_{hotel}, l_{pub} < 1km \\ & station(n''', l_{station}),\ \text{dist}(l_{hotel}, l_{station}) < 1km. \end{aligned}$$

Other examples of distance queries could be real estate agents that are interested in properties that are larger than 1000 square meters and not more than five kilometres from the next town with a school and a supermarket, or evacuation planners, which might want to know which public facilities (e.g. schools, retirement homes, kindergartens) are within a certain range around a chemical industry complex.

Given a distance query, the *component query* for the literal L_i is defined as the query $Q(\bar{t}_i, l_i) :- R_i(\bar{t}_i, l_i)$. In the next section, completeness of component queries will be a central building block for assessing the completeness of distance queries.

4.2 Completeness Definitions

In many scenarios the open-world assumption is employed for databases. The open-world assumption is that a database is not guaranteed to capture all facts from the domain of interest that hold in the real world. This assumption is particularly natural for volunteered data.

Still, such databases may be complete for parts of the real world. This can be expressed using completeness statements.

Definition 1 (Completeness statement). *A completeness statement is a pair $(R(\bar{t}, l), A)$, where R is an object class, $\bar{t}$ is a tuple consisting of constants and the special symbol '$*$', and A is an area. It has an associated simple query Q_C, which is defined as $Q_C(\bar{t}, l) :- R(\bar{t}, l), l \in A$, where the '$*$' are replaced by distinct new variables.*

Example 5. Consider two areas A_1 and A_2 as shown in Fig. 5. A completeness statement c_1 expressing that hotels with four stars are complete in the area A_1 would be written as $Compl(hotel(*, 4, *), A_1)$. A statement c_2 expressing that parks are complete in the area A_2 would be written as $Compl(park(*, *), A_2)$.

The simple query Q_{c_1} corresponding to the statement c_1 would be $Q_{c_1}(n, s, r, l) :- hotel(n, 4, r, l), l \in A_1$.

While under the open-world assumption, in general anything more can hold in reality, completeness statements set

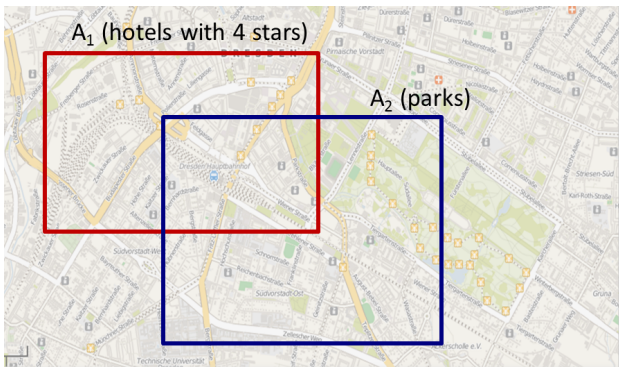

Figure 5: Areas A_1 and A_2.

constraints: They state that in certain parts, the database contains already everything that holds in the real world, therefore, the real world cannot contain any more information in these parts. Completeness statements therefore constrain what the real world can look like.

A database D^i *satisfies* a completeness statement C wrt. a given database D, if the query Q_C does not return more objects over D^i than over D.

Example 6. Consider the database D_{Abgd} from Ex. 2 and consider the completeness statements c_1 and c_2 from Ex. 5. A database D^i that contains an additional hotel Marygold satisfies the completeness statements, as long as the Marygold hotel has either not four stars, or is not located within the area A_1. If the Marygold hotel has four stars and is inside the area A_1, then D^i would violate the completeness statement c_1, because Q_{c_1} would return the additional answer Marygold over D^i, which is not returned over D_{Abgd}.

Definition 2 (Possible Completions). *Given a database D and a set of completeness statements $\mathcal{C}$, a database D^i is called a possible completion for D if*

- *$D \subseteq D^i$ and*
- *D^i satisfies all statements in $\mathcal{C}$ wrt. D.*

We write $Ext_{\mathcal{C}}(D)$ to denote the set of all possible completions for D wrt. $\mathcal{C}$.

Note that for any D and $\mathcal{C}$, the set of possible completions contains at least D again and thus is never empty.

Having defined the possible extensions, we can now define the first goal of the completeness assessment, namely the answer classification.

Definition 3 (Candidate Classification). *Let D be a fixed database instance and $\mathcal{C}$ be a set of completeness statements. Then given a distance query Q, each candidate object o that satisfies the core query Q_{L_0} of Q in D belongs to exactly one of the following categories:*

- *Certain Answers: If o is an answer to Q over all possible completions to wrt. $\mathcal{C}$, that is, $o \in Q(D^i)$ for all D^i in $Ext_{\mathcal{C}}(D)$.*
- *Impossible Answers: If there is no possible completion to D wrt. $\mathcal{C}$ where o is an answer.*
- *Possible Answers: If o is not a certain answer, but there exists at least one D^i in $Ext_{\mathcal{C}}(D)$ such that $o \in Q(D^i)$.*

We denote the sets of certain, impossible and possible answers of Q as $cert_{D,\mathcal{C}}(Q)$, $impos_{D,\mathcal{C}}(Q)$ and $poss_{D,\mathcal{C}}(Q)$, respectively.

In the following, we will usually assume that D and $\mathcal{C}$ are be fixed, and will therefore drop the subscript for the answer categories and also for the completeness area.

Example 7. Consider again the database D_{Abgd} from Ex. 2 and the completeness statements c_1 and c_2 from Ex. 5. The candidate objects for Q, that is, the objects that satisfy the query Q_{L_0}, are the hotels Moonshine Star, Holiday Inn and British Rest.

Of these, intuitively, the Moonshine Star hotel is a certain answer, because it already satisfies the query, so it will also satisfy it for any possible more complete database. The Holiday Inn is an impossible answer, because it currently does not satisfy the query, and, according to the completeness statement c_2 there also cannot appear any parks in the real world that would make it an answer. The British Rest is a possible answer, because currently it does not satisfy the query, but the completeness statements do not exclude the possibility that in the real world there are parks nearby. We will discuss the classification of these hotels again in Ex. 10.

The second task of the completeness assessment for a query is determining the area in which no new answers can appear at all. We call this area the *completeness area* of the query.

Given a query Q for objects, its variant $\tilde{Q}$ outputs only the locations of the objects. We can now define the completeness area of a distance query as follows.

Definition 4 (Completeness Area). *Let Q be a distance query, D be a database and $\mathcal{C}$ be a set of completeness statements. Then the completeness area $\mathrm{CA}_{D,\mathcal{C}}(Q)$ is the maximal area A such that $\tilde{Q}(D) \cap A = \tilde{Q}(D^i) \cap A$ for all possible completions D^i of D wrt. $\mathcal{C}$.*

An example for the completeness area can be seen in Fig. 6.

In the next section we discuss how the answer categories and the completeness area can actually be computed.

5. COMPLETENESS ASSESSMENT

For ease of presentation, we first discuss the assessment for queries that do not contain negated literals, and extend the techniques to queries containing negated literals later.

5.1 Positive Queries

Given a query Q, a priori any object that satisfies the center query Q_{L_0} could become an answer. For positive queries, the identification of the certain answers is easy:

Proposition 5 (Certain Answers). *Let Q be a positive distance query, D be a database and $\mathcal{C}$ be a set of completeness statements. Then:*

$$cert(Q) = Q(D).$$

Note that the computation of certain answers is only so easy because positive queries are monotonic. For queries with negation "$cert(Q) = Q(D)$" does not hold.

To divide the remaining answers to Q_{L_0} into possible and impossible answers, and to compute the completeness area, we need more formalisms. We first analyse how to compute the completeness area for simple queries.

Given two tuples t_1 and t_2 of constants and distinct variables, we say that t_1 *subsumes* t_2 if t_1 has the same constant or a variable at every position where t_2 has a constant.

Example 8. Consider three tuples $(*, *, *)$, $(*, 4, *)$ and $(*, 4, yes)$. Then the first tuple subsumes the latter two, and the second statement subsumes the last one.

Proposition 6 (Completeness Area for Simple Queries). *Let $Q(\bar{t}, l) :- R(\bar{t}, l)$ be a simple query and $\mathcal{C}$ be a set of completeness statements. Then $\mathrm{CA}_{\mathcal{C}}(Q)$, the completeness area of Q wrt $\mathcal{C}$ is computed as follows:*

$$\mathrm{CA}_{\mathcal{C}}(Q) = \bigcup \{A_i \mid t_i \text{ subsumes } t \text{ and } C_i \in \mathcal{C}\}.$$

Observe that this area is independent of database instances.

Example 9. Consider the component query Q_{L_0} of the query $Q_{\text{niceHotels}}$, which is written as $Q_{L_0}(n,4,r,l) \text{:-} \; hotel(n,4,r,l)$. The completeness area for this query is the union of the areas of all completeness statements that subsume the completeness of hotels with four stars, for example statements which talk about the completeness of all hotels.

Given a set of completeness statements $\mathcal{C}$, a database instance D, a literal L and a distance d, we define the area of points that are *certainly out of range*, denoted $Coor_{\mathcal{C},D}(L,d)$, as the set of all points for which in no possible completion an L-object is within distance d. Let $\text{dist}(p,P)$ for a point p and a set of points P be defined as $min\{dist(p,p') \mid p' \in P\}$. Then:

$$Coor_{\mathcal{C},D}(L,d) = \{p \mid \text{dist}(p, Q_L(D^i)) > d \text{ for all } D^i\}$$

The spatial functions *grow* and *shrink* enlarge or downsize geometries by a certain radius. Remember also that for a query Q for objects, its variant $\tilde{Q}$ outputs only the object locations. Using this, the area *Coor* can be computed as follows:

Proposition 7. *Given $\mathcal{C}$, D, L and d as above, it holds:*

$$Coor_{\mathcal{C},D}(L,d) = shrink(\text{CA}(Q_L),d) \setminus grow(\tilde{Q}_L(D),d).$$

Intuitively, this means that in order to compute $Coor_{\mathcal{C},D}(L,d)$, we first identify the completeness area of Q_L and shrink it by the distance d. Consider a point in this shrunken area. Then, this point is certainly out of range d of any L-object, because, due to the shrinking, also no extension of D satisfying $\mathcal{C}$ can contain an L-object within distance d. Next, consider also the set of L-objects present in D. Then no extension of D wrt. $\mathcal{C}$ can contain further L-objects in the completeness area of L. As a consequence the points that are both in the shrunken area *and* have a distance of at least d from the L-objects that are in D and in $\text{CA}(Q_L)$ are certainly out of range of *all* possible L-objects.

We can now compute the possible answers, the impossible answers, and the completeness area as follows.

Theorem 8. *Let Q be a distance query as in Eq. 1 with $n+1$ literals, $\mathcal{C}$ be a set of completeness statements and D be a database instance. Then the following holds:*

(i) $imposs(Q) = Q_{L_0}(D) \cap (Coor_{L_1,d_1} \cup \cdots \cup Coor_{L_n,d_n})$

(ii) $poss(Q) = Q_{L_0}(D) \setminus (cert(Q) \cup imposs(Q))$

(iii) $\text{CA}(Q) = \{\text{CA}_{L_0} \cup Coor_{L_1,d_1} \cup \cdots \cup Coor_{L_n,d_n}\} \setminus poss(Q)$

Proof. (i): Suppose an object o is within some area $Coor_{L_i,d_i}$. By definition of *Coor*, this means (a) that in the current database there is no L_i-object within distance d_i from o, and (b) that the L_i-objects are complete up to distance d_i around o. But this implies that in no possible completion D^i can satisfy the literal L_i of the query, which implies that o is an impossible answer for Q.

(ii): Holds by definition of possible answers.

(iii): We have to show (a) that the query is complete in every point computed by this formula, and (b) that there cannot be any additional points where the query is complete.

Regarding (a), observe that in CA_{L_0} no possible completion can contain further L_0-objects, and that, by the same argument as used for (i), the *Coor*-areas can only contain impossible answers.

Regarding (b), observe that any point p not captured by the above formula is either the location of a possible answer for the query, or both outside CA_{L_0} and outside all the *Coor*-areas for the satellites. If p is the location of a possible answer, this by definition says that there exists a possible extension such that there is an additional answer at point p. If p is outside both CA_{L_0} and all the *Coor*-areas, this means that in some valid extension, an additional L_0-object may occur at p which satisfies the query. □

Example 10. Consider again Mary's query for hotels with a park nearby. The completeness area then would look as shown in Fig. 6, where the rectangular area in the upper left is green, because it is the completeness area for hotels with four stars, and the additional green area to its lower right is green, because both there are no hotels within a distance of two kilometres in the database, nor can there be additional ones in reality because parks are nonexisting and complete in the two-kilometer surrounding.

We can now formally explain the answer categories for the hotels: The Moonshine Star hotel is a certain answer, because it is returned by $Q(D_{Abgd})$. The Holiday Inn is an impossible answer, because it is inside the area $Coor(L_1, 2\,km)$, as it is both inside $shrink(\text{CA}(Q_{L_1}), 2\,km)$ and its distance to the closest park, the King's Garden, is clearly more than two kilometres. The British Rest is a possible answer, because it is not a certain answer but also not in the area $Coor(L_1, 2\,km)$, which means it is not an impossible answer.

Figure 7 shows the relation of the different concepts in completeness assessment. The boxes at the top show the input to the algorithm, the boxes at the bottom the output. The diamonds in the middle are intermediate results. For each box or diamond, the incoming edges represents the concepts needed for computing the concept.

5.2 Queries with Negation

We now look at the reasoning for queries with negation. Recall the query that asks for hotels that do not have a factory nearby. We can observe two things about this query: First, that without knowledge about completeness, there are no certain answers at all, as for any hotel it could be the case that in reality a factory is nearby. Second, that now the completeness area also contains those points where a factory is nearby, as, independent of whether hotels are there or not, it is clear that those hotels will not satisfy the constraint of not having a factory nearby.

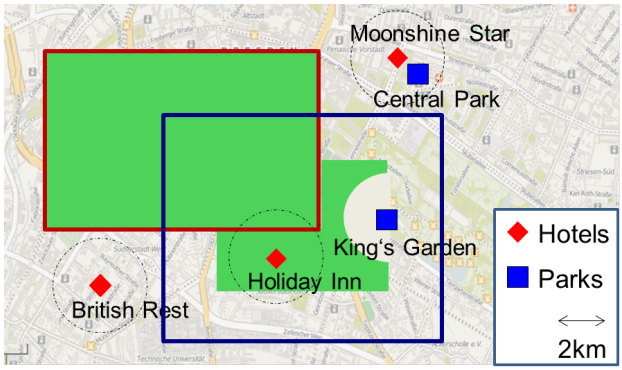

Figure 6: Completeness area (green) for Mary's query. The Moonshine Star is a certain answer, the Holiday Inn an impossible answer, the British Rest a possible answer.

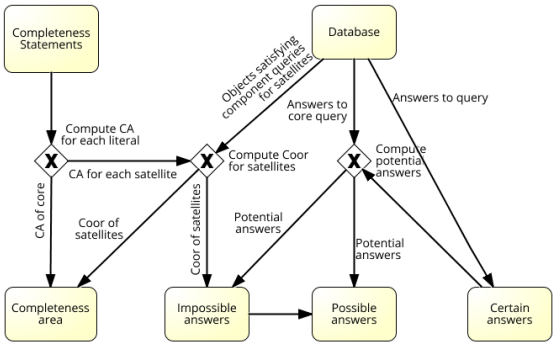

Figure 7: Relation of the concepts in completeness assessment of positive queries.

Formally, a distance query with negation has a form as follows:

$$\begin{aligned} Q(\bar{t}_0, l_0) :- \; & R_0(\bar{t}_0, l_0), \\ & R_1(\bar{t}_1, l_1), \mathrm{dist}(l_0, l_1) < d_1, \\ & \ldots \\ & R_i(\bar{t}_i, l_i), \mathrm{dist}(l_0, l_i) < d_i, \\ & \neg\exists \bar{t}_{i+1}, l_{i+1} : \; R_{i+1}(\bar{t}_{i+1}, l_{i+1}), \mathrm{dist}(l_0, l_{i+1}) < d_{i+1}, \\ & \ldots \\ & \neg\exists \bar{t}_n, l_n : \; R_n(\bar{t}_n, l_n), \mathrm{dist}(l_0, l_n) < d_n \end{aligned}$$

such that literals from 1 to i are positive, and from $i+1$ to n are negated.

To reason about queries with negation, we start with a general observation for distance queries:

Proposition 9 (Universal Query Properties). *Let Q_1 and Q_2 be distance queries asking for objects of the same type. Then also $Q = Q_1 \cap Q_2$ is a distance query, and the following holds:*

(i) $\mathrm{CA}(Q) = \mathrm{CA}(Q_1) \cup \mathrm{CA}(Q_2)$

(ii) $cert(Q) = cert(Q_1) \cap cert(Q_2)$

(iii) $imposs(Q) = imposs(Q_1) \cup imposs(Q_2)$.

For a distance query Q with negation, we define its positive subquery Q^+ as the query $Q^+(\bar{t}_0, l_0) :- L_0, \ldots, L_i$, and its negative subquery Q^- as the query $Q^-(\bar{t}_0, l_0) :- L_0, \neg L_{i+1}, \ldots, \neg L_n$. Clearly, $Q = Q^+ \cap Q^-$.

In Theorem 8 we have already seen how to compute certain and impossible answers for positive queries. To use Proposition 9, we have to show the computation for negative queries.

Similarly to the *Coor* function before, we now introduce an area *Cir* (certainly in range), which is defined as follows:

Given $D, \mathcal{C}$, a literal L and a distance d as before, $Cir_{\mathcal{C},D}(L, d)$ contains all points such that an object satisfying L is within distance d in the database instance D, that is:

$$Cir_{\mathcal{C},D}(L, d) = \{p \mid \mathrm{dist}(p, \tilde{Q}_L(D^i)) < d, \text{ f. a. } D^i\}$$

Because in the ideal databases, information can only be added but never removed, the computation of *Cir* is straightforward:

Proposition 10. *Let $D, \mathcal{C}$, L and d be as before. Then*

$$Cir_{\mathcal{C},D}(L, d) = grow(\tilde{Q}_L(D), d)$$

Having this, we can now compute certain answers, impossible answers and the completeness area for negative distance queries.

Theorem 11. *Let D and $\mathcal{C}$ be as before, and let Q^- be a negative distance query with n literals. Then*

(i) $cert(Q^-) = Q^-(D) \cap Coor(L_1, d_1) \cap \cdots \cap Coor(L_n, d_n)$

(ii) $imposs(Q^-) = Q_{L_0}(D) \cap (Cir(L_1, d_1) \cup \cdots \cup Cir(L_n, d_n))$

(iii) $\mathrm{CA}(Q^-) = \mathrm{CA}(Q_{L_0}) \cup Cir(L_1, d_1) \cup \cdots \cup Cir(L_n, d_n)$.

Due to Prop. 9 and the fact that any distance query with negation can be split into the intersection between a distance query containing only positive satellites and a distance query containing only negated satellites, we now know how to compute the answer categories and the completeness area for distance queries with negation.

5.3 Count Queries

An interesting extension to distance queries are count queries, which are queries that are counting the number of objects satisfying a certain query in a certain area.

For such aggregate queries, one can analyse two things: First, one can calculate the portion of area in which the query is complete, by dividing the intersection between completeness area and query area by the query area (e.g., the query is complete for 80% of Abingdon). Since the completeness area however can contain incomplete points (caused by possible answers), an area which is 100% contained in the completeness area still satisfies query completeness only if the number of possible answers in the area is zero.

Independent of whether the area is 100% complete, we can give bounds for how many of the objects, that satisfy the core in the current database, satisfy the query in reality.

Using the relationship between certain and possible answers, for any valid completion D^i the bounds on the cardinality of answers for the core query that answers for the full query over the completion are:

$$|cert(Q) + poss(Q)| \;\geq\; |Q_{L_0}(D) \cap Q(D^i)| \;\geq\; |cert(Q)|.$$

If the query area is 100% complete, the upper bound is also an upper bound for the whole query answer in reality ($Q(D^i)$).

Example 11. Consider again the database and completeness statements as shown in Figure 5. Then for the query $Q_{\text{niceHotels}}$, the completeness in the green area lies between 50% and 100%, because there is one certain and one possible answer in that area.

5.4 Complexity

We now look into the complexity of computing the completeness area and certain, possible and impossible answers.

The input of the problem are a database D, completeness statements $\mathcal{C}$ and a positive query Q.

When doing completeness assessment, there are two sources of the complexity: The first is the computation of *Coor* for each satellite literal, which is needed twice when computing CA, and once when computing *imposs*.

Computing *Coor* requires evaluation of a simple query ($O(|D|)$ - check for every object whether it satisfies the selection condition of the simple query) and computation of CA ($O(|\mathcal{C}|)$ - check for every completeness statement, whether

talks about objects that are the same or more general than the ones in the literal). Both have to be done also for the core literal, thus, computing *Coor* for all literals has a complexity of $|Q| * (|C| + |D|)$.

The second source of complexity is that for computing the completeness area, we also need to compute the possible answers, and for that the certain answers. Computing the certain answers requires query evaluation, which for distance queries has the complexity $|Q|*|D|^2$, because naively, for every object satisfying the core of Q, the distance to the objects satisfying each satellite has to be calculated.

Adding the complexities of computing the *Coor*-areas and the certain answers, we arrive at an overall complexity as follows:

$$|Q| * (|C| + |D|^2).$$

For the subquery needed to compute the certain answers, standard spatial indexing of the objects will speed up the evaluation. For the retrieval of completeness statements that are relevant for a completeness area CA, indexing on the attributes used most often in completeness statements (e.g. stars for hotels) could be employed.

6. EXPERIMENTS

To show the feasibility of our approach, we have implemented the core reasoning using the Java Topology Suite (JTS), a library that implements spatial object classes and functions. In our experiments, we assume four different object types for which completeness statements can be given. For simplicity, we do not use any constants other than distances in the statements or queries, thus, queries and completeness statements always refer to all objects of a type. Also, the queries are only positive. We place objects and completeness statements randomly in a 1000×1000 space, where completeness statements are rectangles with a random edge length between 1 and 200 (thus, the average statement covers 1% of the space), and queries use random distance constants between 1 and 100. There are three parameters to vary:

- The number of objects (data). The results for this can be seen in Fig. 8 (left).
- The length of the query, measured as number of atoms in its body. The results for this can be seen in Fig. 8 (middle).
- The number of completeness statements (metadata). The results for this can be seen in Fig. 8 (right).

Each time, we compare the runtime for completeness area calculation plus answer classification with the runtime for query evaluation.

As one can see, the reasoning behaves well in terms of the size of the query, and both in terms of the query size and the number of statements, it shows the same asymptotic behaviour as the query execution (linear and quadratic, respectively).

While query evaluation is not affected by the number of completeness statements, completeness reasoning shows again a quadratic behaviour. As discussed in the previous section, spatial indexing techniques for objects and completeness statements could be applied in completeness reasoning analogous as they are used in spatial query evaluation.

7. DISCUSSION

In this section we discuss practical aspects of the presented framework.

How to use completeness information. Once knowing about certain, impossible and possible answers, the question remains what to do with this information. In the example of a tourist, certain answers would be hotels that the tourist might book a room at, possible answer would be hotels where further investigation is needed (check the website of the hotel, call the tourist information, or similar), and impossible answers are the ones that the tourist can ignore. Also the completeness area would tell him for which areas further investigation will be useless.

Depending on the application, the meaning of the completeness area could be inverted. In applications where one is interested to find information that is not yet recorded in the database (treasure hunting illustrates that well, more realistic applications might be e.g. real-estate agents looking for new business opportunities), the complement of the completeness area is the actually interesting area.

Limitations. A limitation of the presented theory is that we have assumed that all objects are points. When objects have an extent, the meaning of completeness statements has to be clarified. Either an object is constrained by a completeness statement, if it partially lies in the area of the completeness statement, or if it lies fully in it. The latter interpretation would however mean that we can be complete for all lakes in the US and in Canada, and could still miss the border lake Lake Ontario.

Also, the reasoning has to take into account the appropriate interpretation of the dist predicate: Whether distances between objects are the minimal distances between their outlines, or the distance between their centers.

Completeness Statements in OpenStreetMap. The statements as used on the OSM Wiki (see Figure 4) are simpler than the ones presented in this paper for two reasons: First, they do not use any constants (there are only statements for all hotels, but not just for hotels with four stars), but instead just state that one out of 12 object classes is complete in a certain area. Second, the areas for which the completeness statements are given do not overlap, instead, the statements are always given for disjoint areas (as opposed to stating that four-star hotels are complete in all Abingdon and hotels with restaurants are complete in the center of Abingdon, which are spatially and semantically overlapping statements).

In OSM, completeness statements come in 7 different levels, ranging from "Unknown" to "Completeness verified by two persons" (see the right table in Fig. 4). In that figure the lower table also contains a row concerning the implications on usage ("Use for navigation"). Still, it remains difficult to see how to interpret the levels and to know the implications on data usage.

So far, the use of completeness statements on the OSM wiki is sparse. More concretely, out of 26,676 pages on the OSM wiki on 30th of June, 2014, only 1,477 (~6%) give completeness statements (estimate based on the number of pages that contain an image from Fig. 4).

Especially, at the moment completeness statements are only given for urban areas. This may change if completeness state-

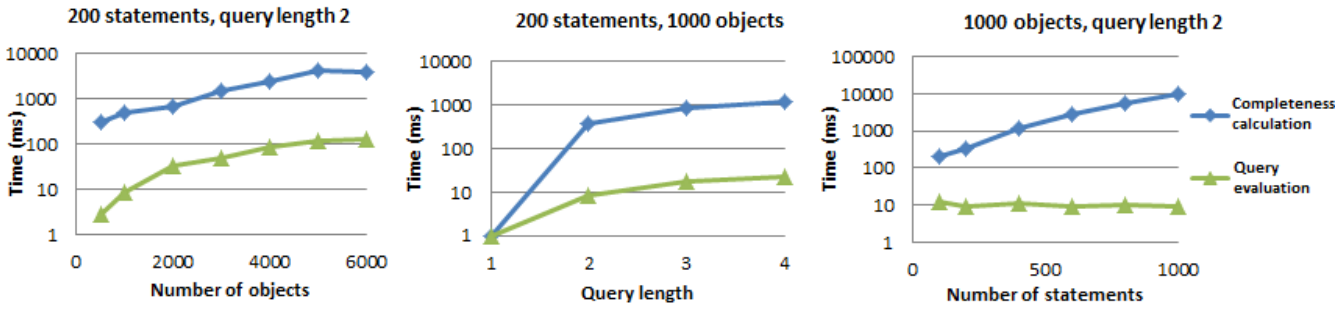

Figure 8: Comparison between query answering and completeness calculation.

ments become more frequently used.

On the technical side a challenge is to make the completeness statements on the OSM wiki machine-readable. The tables that hold the completeness statements are in principle already machine-readable, but the challenge is that at the moment, the areas of the completeness statements are not formalized. The areas that are currently textually described in the second column of the table in Fig. 3 (e.g. "Central + Ock St. to R. Ock") would need to be mapped to spatial objects.

Maintenance of Completeness Statements. A separate challenge is the maintenance of completeness statements. As the real-world changes continuously, new objects can arise that toggle previous completeness statements incorrect, and objects can even disappear.

The first challenge can be addressed by regularly reviewing completeness statement, and giving completeness guarantees only with time stamps ("complete as of xx.yy.zzzz"). The second challenge goes beyond the term of completeness, and instead asks also for correctness guarantees. Mappers then not only would have to guarantee that all information of the real world is captured in the database, but on the contrary also that objects in the database also exist in the real world, and same as for the first challenge, would need to review these statements periodically.

8. CONCLUSION

In this paper we have discussed how to assess the completeness of spatial databases based on metadata. We have introduced the concepts of completeness area, certain, possible and impossible answers for queries. We have then shown how these concepts can be computed for distance queries, using a reduction to assessment of completeness of simple queries.

We have discussed that completeness statements are already present to a limited extend in OpenStreetMap, and have shown that these statements are even simpler than the statements that our framework can handle. We also pointed out the conceptual challenges regarding the maintenance and meaning of completeness statements.

We have also built a demonstrator system, which is available at http://www.inf.unibz.it/~srazniewski/geoCompl/.

Acknowledgement

We are thankful to the user Bigbug21 for information about the OSM community, and to the anonymous Reviewer 1 for very helpful feedback. This work has been partially supported by the project "MAGIC: Managing Completeness of Data" funded by the province of Bozen-Bolzano.

9. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. Foundations of databases. In *Addison-Wesley*, 1995.

[2] R. H. Güting. An introduction to spatial database systems. *VLDB J.*, 3(4):357–399, 1994.

[3] M. Haklay. How good is volunteered geographical information? A comparative study of OpenStreetMap and Ordnance Survey datasets. *Environment and Planning. B, Planning & Design*, 37(4):682, 2010.

[4] M. Haklay and C. Ellul. Completeness in volunteered geographical information—the evolution of OpenStreetMap coverage in England (2008-2009). *Journal of Spatial Information Science*, 2010.

[5] T. Imieliński and W. Lipski, Jr. Incomplete information in relational databases. *J. ACM*, 31:761–791, 1984.

[6] A. Y. Levy. Obtaining complete answers from incomplete databases. In *Proceedings of the International Conference on Very Large Data Bases*, pages 402–412, 1996.

[7] P. Mooney, P. Corcoran, and A. Winstanley. Towards quality metrics for OpenStreetMap. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 514–517. ACM, 2010.

[8] A. Motro. Integrity = Validity + Completeness. *ACM TODS*, 14(4):480–502, 1989.

[9] S. Razniewski and W. Nutt. Completeness of queries over incomplete databases. In *VLDB*, 2011.

[10] S. Razniewski and W. Nutt. Assessing the completeness of geographical data (short paper). In *BNCOD*, 2013.

[11] W. Shi, P. Fisher, and M. Goodchild. *Spatial Data Quality*. CRC, 2002.

[12] T. Wang and J. Wang. Visualisation of spatial data quality for internet and mobile GIS applications. *Journal of Spatial Science*, 49(1):97–107, 2004.

[13] D. Zielstra, H. H. Hochmair, and P. Neis. Assessing the effect of data imports on the completeness of openstreetmap–a united states case study. *Transactions in GIS*, 17(3):315–334, 2013.