Approaches for Semantically Annotating and Discovering Scientific Observational Data*

Huiping Cao[1], Shawn Bowers[2], Mark P. Schildhauer[3]

[1] Dept. of Computer Science, New Mexico State University
hcao@cs.nmsu.edu
[2] Dept. of Computer Science, Gonzaga University
bowers@gonzaga.edu
[3] NCEAS, University of California Santa Barbara
schild@nceas.ucsb.edu

Abstract. Observational data plays a critical role in many scientific disciplines, and scientists are increasingly interested in performing broad-scale analyses by using data collected as part of many smaller scientific studies. However, while these data sets often contain similar types of information, they are typically represented using very different structures and with little semantic information about the data itself, which creates significant challenges for researchers who wish to discover existing data sets based on data semantics (observation and measurement types) and data content (the values of measurements within a data set). We present a formal framework to address these challenges that consists of a semantic observational model, a high-level semantic annotation language, and a declarative query language that allows researchers to express data-discovery queries over heterogeneous (annotated) data sets. To demonstrate the feasibility of our framework, we also present implementation approaches for efficiently answering discovery queries over semantically annotated data sets.

1 Introduction

Accessing and reusing observational data is essential for performing scientific analyses at broad geographic, temporal, and biological scales. Classic examples in earth and environmental science include examining the effects of nitrogen treatments across North American grasslands [17], and studying how changing environmental conditions affect bird migratory patterns [19]. These types of studies often require access to hundreds of data sets collected by independent research groups over many years. Tools that aim to help researchers discover and reuse these data sets must overcome a number of significant challenges: (1) observational data sets exhibit a high level of structural heterogeneity (e.g., see Fig. 1), which includes the use of various terms and conventions for naming columns containing similar or compatible information (e.g., "dw", "wt", "m", "biomass" may each be used to denote a "mass" measurement); and (2) semantic information about data sets, which is crucial for properly interpreting data, is typically either missing or only provided through natural-language descriptions.

* This work supported in part through NSF grants #0743429 and #0753144.

A. Hameurlain et al. (Eds.): DEXA 2011, Part I, LNCS 6860, pp. 526–541, 2011.

site	*plt*	*size*	*ph*	*spp*	*len*	*dbh*
GCE6	A	7	4.5	piru	21.6	36.0
GCE6	B	8	4.8	piru	27.0	45
...	...	...	...	...	...	...
GCE7	A	7	3.7	piru	23.4	39.1
GCE7	B	8	3.9	piru	25.2	42.7
...	...	...	...	...	...	...

yr	*field*	*area*	*acidity*	*piru*	*abba*	...
2005	f1	5	5.1	20.8	14.1	...
2006	f1	5	5.2	21.1	15.2	...
...	...	...	...	...	...	...
2010	f1	5	5.8	22.0	18.9	...
2005	f2	7	4.9	18.9	15.3	...
...	...	...	...	...	...	...

Fig. 1. Typical examples of similar (but not identical) observational data sets consisting of study locations (plot, field), soil acidity measurements, and height and diameter measurements of trees

Despite these challenges, a number of efforts are being developed with the goal of creating and deploying specialized software infrastructures (e.g., [5,1]) to allow researchers to store and access observational data contributed from various disciplines. While a large number of data sets are stored using these repositories, these sites provide primarily simple keyword-based search interfaces, which for many queries are largely ineffective for discovering relevant data sets (in terms of precision and recall) [7].

This paper presents a formal semantic annotation and data discovery framework that can be used to uniformly represent and query heterogeneous observational data. We adopt an approach that is based on a number of emerging observation models (e.g., [3,11,9,15]), which provides canonical representations of observation and measurement structures that researchers can use to help describe, query, and access otherwise heterogeneous data sets. Here we consider the use of description-logic (i.e., OWL-DL) based ontologies for domain-specific terms to specify observation and measurement types that can be used to both annotate data sets and to specify data-discovery queries.

Semantic annotations in our framework define concrete mappings from relational data sets to a uniform observational model specialized by domain-specific terms. The annotation language was designed to support annotations created either manually or automatically (e.g., by employing attribute similarity measures or data-mining techniques). The language is currently being used to store annotations created (via a graphical user interface) within a widely used metadata editing tool [2] for earth and environmental science data sets. A key contribution of our annotation approach is that it provides a declarative, high-level language that follows the "natural" way in which users describe their observational data sets semantically (by focusing on attribute-level metadata, and inferring remaining structural relationships). We also support data-discovery queries posed over both the types of observations and measurements used to annotate data sets as well as over (possibly summarized) values contained within data sets. For instance, using our framework, it is possible to express queries that range from simple "schema-level" filters such as "*Find all data sets that contain height measurements of trees within experimental locations*" to queries that access, summarize, and select results based on the values within data sets such as "*Find all data sets that have trees with a maximum height measurement larger than 20 m within experimental locations having an area smaller than 10 m^2*".

Finally, we describe different storage and query evaluation approaches that have been implemented to support the framework. We consider both a "data warehouse" approach that uses a single "materialized" database to store underlying observational data sets

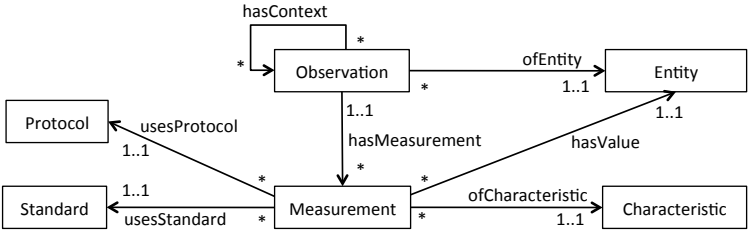


Fig. 2. Main observational modeling constructs used in semantic annotation and data discovery

(where query evaluation involves rewriting a discovery query into a query over the warehouse) and an approach that treats semantic annotations as logical views over the underlying data set schemas (where query evaluation involves rewriting the original query using the annotation into corresponding queries over the underlying data sets). Based on our initial experimental results, we demonstrate the feasibility of querying a large corpus using these approaches, and that querying data in place can lead to better performance compared with more traditional warehousing approaches.

The rest of this paper is organized as follows. In Sec. 2 we present the observational model, semantic annotation language, and data-discovery language used within our framework. In Sec. 3 we describe two implementation approaches. In Sec. 4 we present an initial experimental evaluation. In Sec. 5 we discuss related work, and in Sec. 6 we summarize our contributions.

2 Semantic Annotation and Discovery Framework

Fig. 2 shows the modeling constructs we use to describe and (depending on the implementation) store observational data. An *observation* is made of an *entity* (e.g., biological organisms, geographic locations, or environmental features, among others) and primarily serves to group a set of measurements together to form a single "*observation event*". A *measurement* assigns a value to a *characteristic* of the observed entity (e.g., the height of a tree), where a value is denoted through another entity (which includes primitive values such as integers and strings, similar to pure object-oriented models). Measurements also include *standards* (e.g., units) for relating values across measurements, and can also specify additional information including collection protocols, methods, precision, and accuracy (not all of which are shown in Fig. 2 due to space limitation). An observation (event) can occur within the *context* of zero or more other observations. Context can be viewed as a form of dependency, e.g., an observation of a tree specimen may have been made within a specific geographic location, and the geographic location provides important information for interpreting and comparing tree measurements. In this case, by establishing a context relationship between the tree and location observations, the measured values of the location are assumed to be constant with respect to the measurements of the tree (i.e., the tree measurements are dependent on the location measurements). Context forms a *transitive* relationship among observations. Although not

considered here, we also employ a number of additional structures in the model for representing complex units, characteristics, and named relationships between observations and entities [9]. When describing data sets using the model of Fig. 2, domain-specific entity, characteristic, and standard classes are typically used. That is, our framework allows subclasses of the classes in Fig. 2 to be defined and related, and these terms can then be used when defining semantic annotations.

A key feature of the model is its ability for users to assert properties of entities (as measurement characteristics or contextual relationships) without requiring these properties to be interpreted as *inherently* (i.e., *always*) true of the entity. Depending on the context an entity was observed (or how measurements were performed), its properties may take on different values. For instance, the diameter of a tree changes over time, and the diameter value often depends on the protocol used to obtain the measurement. The observation and measurement structure of Fig. 2 allows RDF-style assertions about entities while allowing for properties to be contextualized (i.e., the same entity can have different values for a characteristic under different contexts), which is a crucial feature for modeling scientific data [9]. Although shown using UML in Fig. 2, the model has been implemented (together with a number of domain extensions) using OWL-DL.[1]

2.1 Semantic Annotation

Semantic annotations are represented using a high-level annotation language (e.g., see Fig. 3), and each annotation consists of two separate parts: (1) a *semantic template* that defines specific observation and measurement types (and their various relationships) for the data set; and (2) a *mapping* from individual attributes of the data set to measurement types defined within the semantic template. The left side of Fig. 3 gives an example annotation for the first table of Fig. 1. Here we define four observation types denoting measurements of sites, plots, soils, and trees, respectively. A site observation contains a simple (so-called "nominal") measurement that gives the name of the site. Similarly, a plot observation records the name of the plot (where a plot is used as an experimental replicate) as well as the plot area. Here, plots are observed within the context of a corresponding site. A soil observation consists of an acidity measurement and is made within the context of a plot observation (although not shown, we would typically label the context relation in this case to denote that the soil is part of the plot). Finally, a tree observation consists of the taxonomic name of the tree along with height and diameter measurements in meters and centimeters, respectively.

The right side of Fig. 3 shows the relationship between (a portion of) the semantic template (top) and an attribute mapping (dashed-lines, middle) from the underlying data set schema (bottom) to the template. As shown, each attribute is assigned to a single measurement type in the template. This approach follows the typical view of attributes in data sets as specifying measurements, where the corresponding entities, observation events, and context relationships are implied by the template. We note that users will not typically specify annotations directly using the syntax shown in Fig. 3. Instead, we have developed a graphical user-interface within [2] that allows users to specify attribute-level mappings to measurement types and the corresponding measurement and

[1] e.g., see `http://ecoinformatics.org/oboe/oboe.1.0/oboe-core.owl`

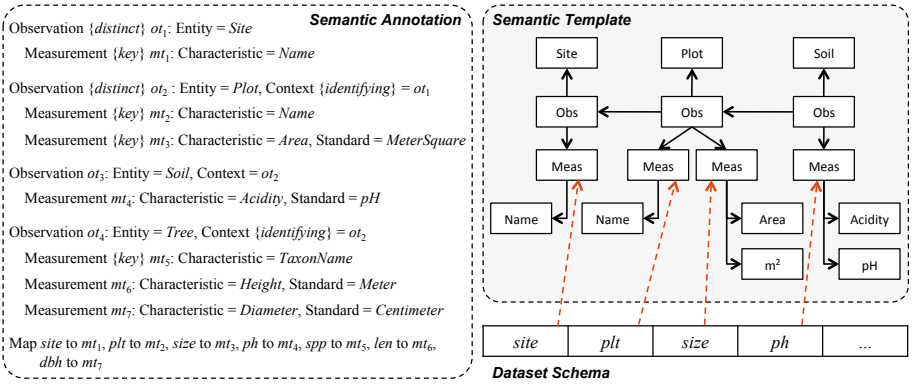


Fig. 3. Semantic annotation of the first data set of Fig. 1 showing the high-level annotation syntax (left) and a portion of the corresponding "semantic template" and schema mapping (right)

observation types of the data set. The annotation language shown here is used to store (via an XML serialization) the mappings and semantic templates generated by the tool.

The meaning of a semantic annotation can be viewed as the result of processing a data set row-by-row such that each row creates a valid instance of the semantic template. For example, in the first row of the data set, the site value "GCE6" implies: (1) an instance m_1 of the measurement type mt_1 whose value is "GCE6"; (2) an instance c_1 of the Name characteristic for m_1; (3) an instance o_1 of an observation (corresponding to type ot_1) having measurement m_1; and (4) an instance e_1 of the Site entity such that e_1 is the entity of o_1. Similarly, assuming the plot attribute value "A" of the first row corresponds to an observation instance o_2 (of observation type ot_2), the context definition for ot_2 results in o_2 having o_1 as context.

The "key", "identifying", and "distinct" constraints are used to further specify the structure of semantic-template instances. These constraints are similar to key and weak-entity constraints used within ER models. If a measurement type is defined as a *key* (e.g., mt_1 in Fig. 3), then the values of instances for these measurement types identify the corresponding observation entity. For example, both the first and second row of the first table in Fig. 1 have a site value of "GCE6". Thus, if the observation instance for the site attribute in the second row of the table is o_3, then the key constraint of mt_1 requires that e_1 be an entity of o_3 where e_1 is the entity instance of the first-row's corresponding observation. An *identifying* constraint requires the identity of one observation's entity to depend (through context) on the identity of another observation's entity. In our example, plot names are unique only within a corresponding site. Thus, a plot with the name "A" in one site is not the same plot as a plot with the name "A" in a different site. Identifying constraints define that the identity of an observation's entity is determined by both its own key measurements and its identifying observations' key measurements. Thus, each site name determines the entity observed through the key constraint, whereas, each plot name determines the plot entity with respect to both its name and its corresponding site (as given by the identifying constraint on the context relationship). The "distinct" constraint on observations is similar to the "key" constraint on measurements, except that it is used to uniquely identify observations (as opposed to observation entities). In

Fig. 3, each row with the same value for the site attribute maps not only to the same observed entity (via the key constraint) but also to the same observation instance (via the distinct constraint). A distinct constraint can only be used if each measurement of the observation is constrained to be a key.

More formally, we can *materialize* semantic annotations as follows. First, we represent sets of annotations using the following relations.

- *Annot*(a, d) states that a is an annotation of data set with id d.
- *ObsType*(a, ot, et, *isDistinct*) states that ot is an observation type in annotation a, has entity type et, and whether it distinct.
- *MeasType*(a, mt, ot, ct, st, ..., *isKey*) states that mt is a measurement type in a, is for observation type ot, and has characteristic type ct, standard type st, etc., and whether mt is defined as a key.
- *ContextType*(a, ot, ot', *isId*) states that observation type ot' is a context type of observation type ot in a, and whether the context relationship is identifying.
- *Map*(a, $attr$, mt, ϕ, v) states that data set attribute $attr$ is mapped to measurement type mt in a, where ϕ is an optional condition specifying whether the mapping applies (based on the values of attributes within the data set) and v is an optional value to use for the measurement (instead of the data set value).

We use the following relations to represent instances of semantic templates.

- *Entity*(d, e, et) states that entity e in data set d is an instance of entity type et.
- *Obs*(d, o, ot, e) states that observation o in data set d is of type ot and is an observation of entity e.
- *Meas*(d, m, mt, v, o) states that measurement m in d is of measurement type mt, has the value v, and is a measurement for observation o.
- *Context*(d, o, o') states that observation o is within the context of o' in d.

We can then evaluate the mapping defined by a semantic annotation a over a data set d using the following algorithm, which results in populating the above relations for template instances.

Algorithm MaterializeDB(a, d)

1). $EntityIndex = \emptyset$; // an index of the form $\{\langle ot, keyvals\rangle \rightarrow e\}$
2). **for each** $row = \langle attr_1, attr_2, \cdots, attr_n\rangle \in d$
3). $MeasSet =$ CreateMeasurements(a, row);
4). // partition measurements based on observation types
5). $MeasIndex =$ PartitionMeasurements(a, $MeasSet$); // returns index $\{ot \rightarrow \{m\}\}$
6). $ObsIndex = \emptyset$; // an index of the form $\{ot \rightarrow o\}$
7). **for each** $ot \rightarrow \{m\} \in MeasIndex$
8). $e =$ CreateEntity(a, ot, $\{m\}$, $EntityIndex$); // updates $EntityIndex$
9). CreateObservation(a, ot, e, $ObsIndex$); // updates $ObsIndex$
10). ConnectContext(a, $ObsIndex$);

As shown, while processing each row we create measurement instances for each mapped attribute (cell) in the row (Line 3), link them to their related observation instances (Line 5–9), and then create proper context links between observation instances (Line 10). The *EntityIndex* is used to ensure only unique entities are created within the data set (based on the values for measurements having a key constraint). Thus, before an entity instance

is created (Line 8), the CreateEntity function first checks using the index if the entity has already been created from a previous row. The CreateMeasurements, CreateObservation, and ConnectContext functions are straightforward and each use the annotation's semantic template to create and connect the corresponding instances. This algorithm runs in $O(n \log m)$ time where n is the number of rows in a data set and m ($\ll n$) is the number of distinct keys within the data set. The algorithm uses $O(nc)$ space where c is the number of columns in the data set (thus, nc is the total number of cells).

The semantic annotation language can easily be expressed using standard schema mapping approaches [16], i.e., annotations have a straightforward reduction to source-to-target tuple-generating dependencies and target equality-generating dependencies. A source-to-target tuple-generating dependency (*st-tdg*) is a first-order formula of the form $\forall \bar{x}(\varphi(\bar{x}) \rightarrow \exists \bar{y}\, \psi(\bar{x}, \bar{y}))$ where $\varphi(\bar{x})$ and $\psi(\bar{x}, \bar{y})$ are conjunctions of relational atoms over source and target schemas, respectively, and $\bar{x}$ and $\bar{y}$ are tuples of variables. We can use st-tdgs to define instances of semantic templates, e.g., the following rule maps the first attribute in the data set of Fig. 3 to measurement type mt_1, where R is used as the name of the data set relation:

$$\forall \bar{x}(R(\bar{x}) \rightarrow \exists \bar{y}\, \mathrm{Meas}(\mathtt{d}, y_1, \mathtt{mt}_1, x_1, y_2) \wedge \mathrm{Obs}(\mathtt{d}, y_2, \mathtt{ot}_1, y_3) \wedge \mathrm{Entity}(\mathtt{d}, y_3, \mathtt{Site}))$$

Here we assume x_1 is the first variable in $\bar{x}$, each y_i in the rule is a variable of $\bar{y}$, and $\mathtt{d}$, $\mathtt{mt}_1$, $\mathtt{ot}_1$, and $\mathtt{Site}$ are constants. A target equality-generating dependency (*t-egd*) takes the form $\forall \bar{y}(\phi(\bar{y}) \rightarrow u = v)$ where $\phi(\bar{y})$ is a conjunction of relational atoms over the target schema and u, v are variables in $\bar{y}$. We can use t-egds to represent key, identifying, and distinct constraints, e.g., the following rule can be used to express the key constraint on measurement type mt_1:

$$\begin{aligned}\forall \bar{y}(&\mathrm{Meas}(\mathtt{d}, m_1, \mathtt{mt}_1, v, o_1) \wedge \mathrm{Obs}(\mathtt{d}, o_1, \mathtt{ot}_1, e_1) \wedge \mathrm{Meas}(\mathtt{d}, m_2, \mathtt{mt}_1, v, o_2) \\ &\wedge \mathrm{Obs}(\mathtt{d}, o_2, \mathtt{ot}_1, e_2) \rightarrow e_1 = e_2)\end{aligned}$$

The annotation language we employ is also similar to a number of other high-level mapping languages used for data exchange (e.g., [10,6]), but supports simple type associations to attributes (e.g., as shown by the red arrows on the right of Fig. 3) while providing well-defined and unambiguous mappings from data sets to the observation and measurement schema.

2.2 Data Discovery Queries

Data discovery queries can be used to select relevant data sets based on their observation and measurement types and values. A *basic discovery query* Q takes the form

$$Q ::= \mathtt{EntityType}(\mathtt{Condition})$$

where `EntityType` is a specific observation entity class and `Condition` is a conjunction or disjunction of zero or more conditions of the form

$$\begin{aligned}\mathtt{Condition} ::= \;& \mathtt{CharType}\ [\ \mathtt{op}\ \mathtt{value}\ [\ \mathtt{StandardType}\ ]\ ] \\ \mid\; & f(\mathtt{CharType})\ [\ \mathtt{op}\ \mathtt{value}\ [\ \mathtt{StandardType}\ ]\ ] \\ \mid\; & \mathtt{count}(\ [\ \mathtt{distinct}\ ]\ *)\ \mathtt{op}\ \mathtt{value}\end{aligned}$$

Square brackets above denote optional components and f denotes a standard aggregation function (i.e., sum, avg, min, or max). A data set is returned by a basic discovery query if it contains observations of the given entity type that satisfy the corresponding conditions. We consider three basic types of conditions (for the three syntax rules above): (1) the observation must contain at least one measurement of a given characteristic type (`CharType`) with a measured value satisfying a relational (i.e., $=$, $\neq$, $>$, $<$, $\geq$, $\leq$) or string comparison (e.g., `contains`); (2) the aggregate function applied to measurements of the characteristic type (`CharType`) for all observations of the entity type must satisfy the relational comparison; and (3) the number (`count`) of all observations of the entity type must satisfy the relational comparison (where `distinct` restricts the set of observations to those within unique entities). For instance, in the following basic discovery queries

Tree(TaxonName = 'piru')
Tree(TaxonName = 'piru' $\wedge$ count(distinct $*$) ≥ 5)

the first query select data sets with at least one Tree observation labeled as having the (abbreviated) taxon name "piru", and the second query restricts the returned data sets of the first query to contain at least five such observations.

A *contextualized discovery query* generalizes basic discovery queries to allow selections on context. A contextualized query Q_C for $n \geq 1$ has the form

$$Q_C \ ::= \ Q_1 \rightarrow Q_2 \ \cdots \rightarrow Q_n$$

where each Q_i is a basic discovery query and $\rightarrow$ denotes a context relationship. In particular, a data set satisfies a contextualized query if it satisfies each basic query Q_i and each matching Q_i is related by the given context constraint. To illustrate, the following examples can be used to express the two queries of Sec. 1:

Tree(Height) $\rightarrow$ Plot()
Tree(max(height) $\geq$ 20 Meter) $\rightarrow$ Plot(area $<$ 10 MeterSquared).

That is, the first query returns data sets that contain height measurements of trees within plots (experimental locations), and the second returns data sets that have trees of a maximum height larger than 20 m within plots having an area smaller than 10 m^2. For a collection of data sets D and a contextualized discovery query Q, in the normal way, we write $Q(D)$ to denote the subset of data sets in D that satisfy Q. Note that $Q(D)$ can be computed on a per data set basis, i.e., by checking each data set in D individually to see whether it satisfies Q.

3 Implementation Strategies

In this section we describe two different strategies for evaluating data discovery queries over annotated observational data sets, as shown in Fig. 4. Both strategies utilize semantic annotations (and corresponding ontologies) to answer discovery queries. We assume annotations are stored using the relations described in Sec. 2.1, namely the *Annot*, *ObsType*, *MeasType*, *ContextType*, and *Map* tables. The two approaches differ in how they utilize different representations of the underlying data sets. The first query strategy (*rdb*)

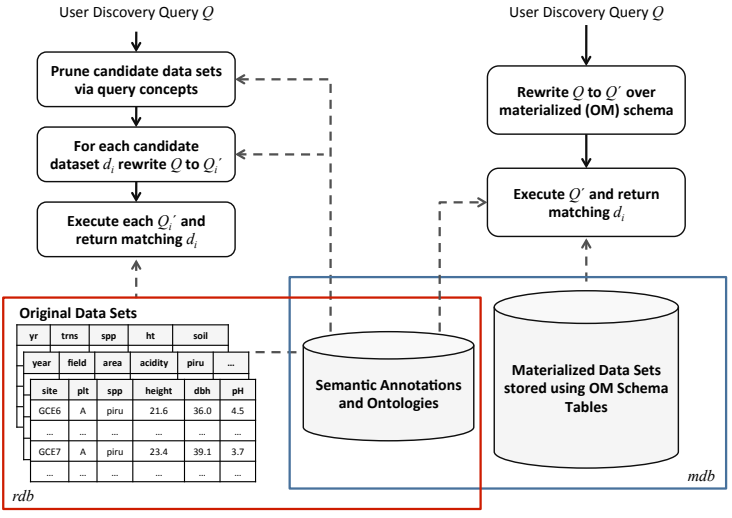


Fig. 4. Different strategies for answering a data discovery query Q: the first strategy stores each data set in its "raw" form (left), and the second strategy materializes each data set into the a common schema (right).

stores each data set in its "raw" form (i.e., "in place") according to its defined schema. Thus, each data set is stored as a distinct relation in the database. For instance, both data sets of Fig. 1 would be stored as tables without any changes to their schemas. The second query strategy (*mdb*) materializes each data set using the *MaterializeDB* algorithm. In this approach, the observations and measurements stored in each data set are represented using the *Entity*, *Obs*, *Meas*, and *Context* tables described in Sec. 2.1.

3.1 Query Evaluation over In-Place Database

Evaluating a discovery query Q in the *rdb* approach consists of three steps. Here we assume that each data set (denoted by id d) is stored in a relation R_d. The first step prunes the search space of candidate data sets to select only those data sets with the required entity, characteristic, and standard types specified within Q. The candidate data sets (i.e., those that potentially match Q) are selected by accessing the semantic-annotation relations $ObsType$, $MeasType$, and $Annot$. This pruning step can help decrease the cost of evaluating discovery queries by reducing the number of data sets whose values must be accessed (for those queries that select data sets based on data values). The second step translates Q into an SQL query Q' over each candidate data set relation R_d. After the first step, we obtain the measurement types (*mt*) related to the discovery query Q (via the *MeasType* relation). We then find the attributes $attr_q$ in each candidate relation R_d via the Map relation (which contains correspondences between attributes $attr$ and measurement types mt). If a basic query does not have any aggregations, these attributes are enough to form the resulting SQL over R_d. Otherwise, we must obtain the key measurement types $attr_{key}$ for the observation types of Q. The key measurement types of an observation type consist of its own key measurement types

and those of its identifying observation types, which must be retrieved by traversing the identifying context chain (i.e., by searching the $ObsType$ and $ContextType$ relations). Once the needed attributes for each candidate data set d are obtained, Q is translated into the following SQL query (where square brackets denote optional clauses).

```
SELECT DISTINCT d[, attr_key] FROM R_d
WHERE non-aggregation conditions
[GROUP BY attr_key]
[HAVING aggregation condition];
```

We illustrate the rewriting process with the following example. Consider the semantic annotations in Fig. 3 and the basic discovery query

Tree(TaxonName = 'piru' $\wedge$ count(distinct $*$) ≥ 5)

from Sec. 2.2. The first step is to find the attributes involved in the condition (i.e., TaxonName = 'piru'). From the entity type "Tree" and the characteristic "TaxonName", we can find the corresponding observation type "ot_4", measurement type "mt_5", and the attribute "spp". We then find the key attributes to perform the aggregation. The key measurement types for entity type "Tree", whose observation type is ot_4, come from ot_4 and ot_4's identifying observation types ot_2 and ot_1. Since these observation types' key measurement types are mt_5, mt_3, mt_2 and mt_1, the key attributes $attr_{key}$ are *spp*, $size$, plt, and $site$ (from the Map relation). The resulting SQL query is expressed as follows where the data set id is denoted d_1 and corresponding relation is denoted R_{d1}.

```
SELECT DISTINCT d1 FROM R_d1
WHERE spp = 'piru'
GROUP BY d1, spp, size, plt, site
HAVING count(*) ≥ 5
```

Finally, in the third step of the *rdb* approach, we execute the SQL query for each of the candidate data sets such that the answer for Q is the union of these results.

Cost analysis. The major computation cost using the *rdb* approach is to send multiple SQL queries to the database server to search the needed information for all the different candidate data tables. Thus, the cost increases with larger numbers of candidate data sets.

3.2 Query Evaluation over Materialized Database

The second query strategy (*mdb*) evaluates a given discovery query Q over the materialized database by directly rewriting Q into an SQL query expressed over the annotation and instance relations of Sec. 2.1. This approach differs from *rdb*, which requires obtaining the underlying attributes for *each* individual candidate table and where one SQL query is constructed per candidate table. Instead, using *mdb*, a single SQL query is created to answer the entire basic data discovery query.

The way in which a basic query is rewritten depends on whether it has an aggregation condition. For a basic query without an aggregation condition, we access the $Entity$ and Obs relations to check the entity conditions, and search the *MeasType* and *Meas* tables for characteristic and standard conditions. For a basic discovery query with an aggregation condition, we perform the aggregation by grouping $Entity.e$ or $Obs.o$

depending on whether the aggregation is on a distinct entity instance or observation instance. Thus, a discovery query Q can be re-written to an SQL query Q' using the *mdb* approach as follows.

```
SELECT DISTINCT Annot.d [, Entity.e][, Obs.o]
FROM Annot, Entity, Obs, MeasType, Meas
[WHERE table join condition [AND selection condition]]
[GROUP BY Annot.d[, Entity.e][, Obs.o] ]
[HAVING aggregation condition ];
```

where:

```
table join condition= (Annot.a = MeasType.a) AND (Annot.d = Meas.d)
  AND (MeasType.mt=Meas.mt) AND(Meas.o=Obs.o) AND (Obs.e=Entity.e)
```

If a basic query contains multiple measurement conditions (of the same entity type), the corresponding SQL query must be combined using "INTERSECTION" or "UNION" operations to answer the basic query of one entity type. For example, the query

Tree(TaxonName = 'piru' $\wedge$ count(distinct $*$) ≥ 5)

is rewritten using the *mdb* approach as:

```
SELECT DISTINCT Annot.d
FROM Annot, Entity, Obs, MeasType, Meas
WHERE table join condition
    AND MeasType.ct='TaxonName' AND Meas.v = 'piru'
GROUP BY Annot.d, Obs.o
HAVING COUNT(*)>5;
```

Cost analysis: The major computation cost in *mdb* involves the cost of joining over the type and instance relations, and the selection cost over the measurement values.

De-normalized materialized database. The join condition shows that a large portion of the cost comes from the join operation over the measurement instance, observation instance, and entity instance relations. To reduce the join cost, we can de-normalize the instance relations $Entity$, Obs, and $Meas$ into a single relation. This strategy will use slightly more space, but can improve the performance of query evaluation for *mdb*.

Horizontally partitioned database. When all the data values are placed into a single measurement table, their data types must be of the same type. This requires using type casting functions provided by the database system to perform type conversion for evaluating queries with algebraic or aggregation operators. This incurs a full scan of the $Meas$ table regardless of whether there is an index on the value columns or not. In our current implementation, we address this issue by partitioning the measurement instance table according to the different data types (e.g., numeric, char, etc.). This partitioning does not incur additional space overhead.

3.3 Executing Complex Discovery Queries

To evaluate more complex data-discovery queries that consist of context relationships, or conjunctions and disjunctions of basic queries (with different entity types), we decompose the query into query blocks and then combine the results of each decomposed

query block. In the first step of query decomposition, a complex query is reformulated into disjunctive normal form (DNF) such that each DNF component is either (1) a basic data discovery query, (2) a conjunction of basic discovery queries with different entity types, or (3) a contextualized discovery query. In the second step of implementing and integrating the decomposed query blocks, we propose two approaches, ExeD and ExeH.

ExeD: Executing a query block based on Decomposed query units. In ExeD, each DNF is further *decomposed* into basic discovery query components. Each of these most decomposed query units is rewritten and executed using one of the strategies discussed in the above two sections. Then, the result of each such most decomposed component is combined (outside of the DBMS) to obtain the result of every DNF. In particular, for the first case where the DNF clause is a basic discovery query, the algorithm simply executes this basic query. For the second case where the DNF component is a conjunction of basic discovery queries, the algorithm intersects the results of the further decomposed components in this DNF. For the third case of contextualized discovery queries, the algorithm runs the basic queries and their context queries and intersects the results. Finally, the results of different DNF components are unioned as the final result.

ExeH: Executing a query block based on Holistic sub-queries. The ExeD approach may incur unnecessarily repeated scans of the database since it evaluates each decomposed basic unit using the DBMS and combines the results externally, outside of the system. For instance, consider the query

$$\text{Tree(max(height)} \geq 20 \text{ Meter)} \rightarrow \text{Plot(area} < 10 \text{ MeterSquared)}.$$

Using the ExeD approach, we need to send two basic queries, Tree(max(height) $\geq$ 20 Meter) and Plot(area $<$ 10 MeterSquared), to the same table. Instead, we form a "holistic" SQL query for each possible basic query block. This holistic SQL is then executed by taking advantage of the optimization capabilities of the DBMS.

Note that not every complex query can be rewritten to one holistic SQL query. Specifically, queries with aggregations must be performed by grouping key measurements. When we have discovery queries with multiple aggregation operations, the *group by* attributes for each aggregation may not be the same. In ExeH, we categorize query blocks into those with and without aggregations. All the query blocks without aggregation conditions are combined and rewritten into one holistic SQL query, while the query blocks with aggregations are processed individually.

4 Experimental Evaluation

In this section we describe the results of our experimental evaluation of the framework and algorithms discussed above. Our implementation was written in Java, and all experiments were run using an iMac with a 2.66G Intel processor and 4G virtual memory. We used PostgreSQL 8.4 as the back-end database system. To report stable results, all numbers in our figures represent the average result of 10 different runs (materialization tasks or queries) with the same settings for each case.

Data. We generated synthetic data to simulate a number of real data sets used within the Santa Barbara Coastal (SBC) Long Term Ecological Research (LTER) project [4].

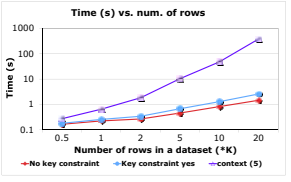


Fig. 5. Data materialization

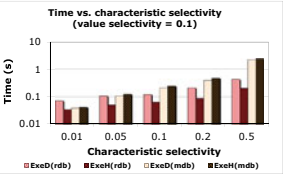


Fig. 6. Time vs. cha. selectivity

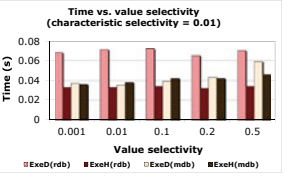
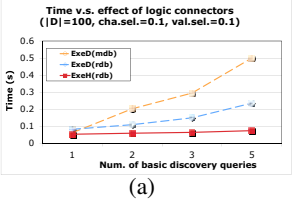
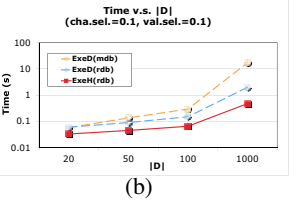


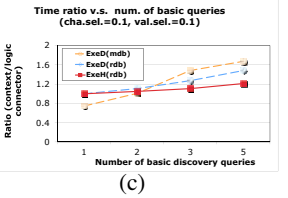
Fig. 7. Time vs. val. selectivity



(a)



(b)



(c)

Fig. 8. Scalability test on (a) the number of logic connectors (e.g., AND and OR), (b) data set size (with three basic queries connected by the logic connector)), and (c) length of context chains (compared with logic connectors). (ExeH(mdb) is not included because it shows similar results to ExeD(mdb).)

This repository contains ∼130 data sets where each one has 1K to 10K rows and on average 15 to 20 columns. To simulate this repository (to test scalability, etc.), our data generator used the following parameters. The average number of attributes and records in a data set is 20 and 5K respectively. The average number of characteristics for an entity is two. The distinctive factor $f \in (0, 1]$, which represents the ratio of distinct entity/observation instances in a data set, is set to 0.5. Our synthetic data generator also controls the *attribute selectivity* to facilitate the test of query selectivity. In particular, given a selectivity $s \in (0, 1]$ of an attribute $attr$, a repository with $|D|$ data sets will be generated to have $|D| \cdot s$ data sets with attribute $attr$.

Queries. Test queries were generated with controlled characteristic selectivity and value selectivity, where the characteristic selectivity determines the percentage of data sets that satisfy a given query and the value selectivity determines the percentage of data rows in a data set satisfying the query. We omit the details of query generation due to space limitations.

Test 1: Materializing data. We first tested the efficiency of the materialization method using data sets generated with distinctive factor $f = 0.5$, number of columns 20, and various number of rows. The annotations over these data sets are the same on observation and measurement types. They differ in the key constraints. "No key constraint" and "Key constraint yes" refer to cases where either no key or key constraints exist in the semantic annotations of data sets; both of these two cases do not include any context constraints. The "context (5)" represents data sets that are semantically annotated with a context chain of 5 observation types (with implicit key constraints).

Fig. 5 shows that the materialization time for each case (every line) is linear to the number of rows in these data sets. This is consistent with our analysis in Sec. 2.1. The

"no key constraint" uses the least amount of time because it does not need to do any additional computation to enforce the uniqueness of the entity and observation instances. The "context (5)" uses the most amount of time because of the context materialization.

Test 2: Querying databases. We also tested the effectiveness and efficiency of our two query strategies: query evaluation using rdb and mdb by utilizing *ExeD* and *ExeH*. Test 2.1 examines how the different query strategies are affected by *value selectivity and characteristic selectivity*. For this test, we used a data repository with 100 data sets and evaluated queries consisting of two basic discovery units connected with the "AND" logic connector. Fig. 6 shows that the execution time of different query strategies increases with the increase of the characteristic selectivity. For *rdb*, higher characteristic selectivity means that more candidate tables are involved in answering a query, thus more SQL queries (against these candidates) are executed. For *mdb*, higher selectivity involves more materialized instances in the join condition, thus more time is used. When we fix the characteristic selectivity and vary the value selectivity, we can see that the execution time is almost constant (Fig. 7) for *rdb* because the number of candidate data tables is the same. For *mdb*, the execution time grows slightly with the increase of the value selectivity also due to the increase in the number of materialized instances. These two figures show that, with smaller characteristic selectivity (e.g., 0.01), the query strategy over *mdb* performs better than for *rdb*. But with larger characteristic selectivities (e.g., 0.5), query strategies over *rdb* perform better. This is because queries over larger amounts of materialized instances, which is the case for larger characteristic selectivities, take more time to perform joins, compared with executing SQL queries over individual candidate data tables in *rdb*.

Test 2.2 focuses on the scalability of the different query strategies. Fig. 8(a) illustrates that the three methods grow linearly to the number of logic connectors. When the number of logic connectors grow, *ExeD(mdb)* grows much faster than *Exe(rdb)*. This is because every basic query in the complex query needs to access the large number of instance tables once for *mdb*. While using *rdb*, when the number of logic connectors is large, *ExeH(rdb)* is still almost constant because the times required to scan the database are almost the same. Fig. 8(b) shows that these different approaches grow linearly to the size of data sets $|D|$ when the value selectivity and characteristic selectivity are fixed. Fig. 8(c) plots the ratio of the execution time of performing a complex query with logic connectors and that of performing a contextualized query with the same number of basic query units. All three methods grow linearly to the number of basic queries. However, the query over *rdb* grows slower than the *mdb* since the latter needs to access the context instance table. As these results show, rewriting queries to the underlying data set schemas outperforms the materialization approach (i.e., the standard warehousing approach) with respect to both the cost of storage (especially due to de-normalization and executing the materialization algorithm) and overall query execution time.

5 Related Work

Data management systems are increasingly employing annotations to help improve search (e.g., [12,18,8]). For example, MONDRIAN [12] employs an annotation model and query operators to manipulate both data and annotations. However, users must be

familiar with the underlying data structures (schemas) to take advantage of these operators, which is generally not feasible for observational data in which data sets exhibit a high degree of structural and semantic heterogeneity. Efforts have also been carried out for leveraging annotations, e.g., for the discovery of domain-specific data [13,20]. These approaches are largely based on keyword queries, and do not consider structured searches. Our work differs from these approaches in that we consider a highly structured and generic model for annotations with the aim of providing a uniform approach for issuing structured data-discovery searches. Our work is closely aligned to traditional data integration approaches (e.g., [14,16]), where a global mediated schema is used to (physically or logically) merge the structures of heterogeneous data sources using mapping constraints among the source and target schemas. As such, the observational model we employ in our framework can be viewed as a (general-purpose) mediation schema for observational data sets. This schema can be augmented with logic rules (as target constraints) and uses the semantic annotations as mapping constraints. However, instead of users specifying logic constraints directly, we provide a high-level annotation language that simplifies the specification of mappings and more naturally aligns with the observation model. In addition, our work focuses on implementing practical approaches for rewriting and optimizing queries (that can include aggregation and summarization operators) over our annotation approach. In particular, our goal is to create a feasible, scalable, and deployable system for applying these approaches for data discovery and exploratory data analysis within existing scientific data repositories.

6 Conclusion

We have presented a novel framework for querying observational data based on formal semantic annotations and a data discovery language that allows structural queries over both schema and data. We also have considered different strategies for efficiently implementing the framework. Our approach involves different forms of query rewriting over annotations and data set schemas. We also have examined the effect of different storage schemas on the query strategies. Our experiments show that in most cases answering queries "in place" outperforms more traditional warehouse-based approaches. As future work we intend to continue to investigate approaches for optimization including the use of indexing schemes and their use in query rewriting approaches.

References

1. Knowledge network for biocomplexity (KNB), `http://knb.ecoinformatics.org`
2. Morpho metadata editor, `http://knb.ecoinformatics.org`
3. OpenGIS: Observations and measurements encoding standard (O&M), `http://www.opengeospatial.org/standards/om`
4. Santa Barbara Coastal LTER repository, `http://sbc.lternet.edu/data`
5. The Digital Archaeological Record (tDAR), `http://www.tdar.org`
6. An, Y., Mylopoulos, J., Borgida, A.: Building semantic mappings from databases to ontologies. In: AAAI (2006)
7. Berkley, C., et al.: Improving data discovery for metadata repositories through semantic search. In: CISIS, pp. 1152–1159 (2009)

8. Bhagwat, D., Chiticariu, L., Tan, W.C., Vijayvargiya, G.: An annotation management system for relational databases. In: VLDB (2004)
9. Bowers, S., Madin, J.S., Schildhauer, M.P.: A conceptual modeling framework for expressing observational data semantics. In: Li, Q., Spaccapietra, S., Yu, E., Olivé, A. (eds.) ER 2008. LNCS, vol. 5231, pp. 41–54. Springer, Heidelberg (2008)
10. Fagin, R., Haas, L.M., Hernández, M., Miller, R.J., Popa, L., Velegrakis, Y.: Clio: Schema mapping creation and data exchange. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 198–236. Springer, Heidelberg (2009)
11. Fox, P., et al.: Ontology-supported scientific data frameworks: The virtual solar-terrestrial observatory experience. Computers & Geosciences 35(4), 724–738 (2009)
12. Geerts, F., Kementsietsidis, A., Milano, D.: Mondrian: Annotating and querying databases through colors and blocks. In: ICDE, p. 82 (2006)
13. Güntsc, A., et al.: Effectively searching specimen and observation data with TOQE, the thesaurus optimized query expander. Biodiversity Informatics 6, 53–58 (2009)
14. Halevy, A., Rajaraman, A., Ordille, J.: Data integration: the teenage years. In: VLDB 2006 (2006)
15. Balhoff, J., et al.: Phenex: Ontological annotation of phenotypic diversity. PLoS ONE 5 (2010)
16. Kolaitis, P.G.: Schema mappings, data exchange, and metadata management. In: PODS 2005 (2005)
17. Pennings, S., et al.: Do individual plant species show predictable responses to nitrogen addition across multiple experiments? Oikos 110(3), 547–555 (2005)
18. Reeve, L., Han, H.: Survey of semantic annotation platforms. In: SAC 2005 (2005)
19. Sorokina, D., et al.: Detecting and interpreting variable interactions in observational ornithology data. In: ICDM Workshops, pp. 64–69 (2009)
20. Stoyanovich, J., Mee, W., Ross, K.A.: Semantic ranking and result visualization for life sciences publications. In: ICDE, pp. 860–871 (2010)