Declarative Cartography: In-Database Map Generalization of Geospatial Datasets

Pimin Konstantin Kefaloukos [#,*,+], Marcos Vaz Salles [#], Martin Zachariasen [#]

[#] *University of Copenhagen* Copenhagen, Denmark
[*] *Grontmij A/S* Glostrup, Denmark
[+] *Geodata Agency (GST)* Copenhagen, Denmark
{kostas, vmarcos, martinz}@diku.dk

***Abstract*—Creating good maps is the challenge of map generalization. An important generalization method is selecting subsets of the data to be shown at different zoom-levels of a zoomable map, subject to a set of spatial constraints. Applying these constraints serves the dual purpose of increasing the information quality of the map and improving the performance of data transfer and rendering. Unfortunately, with current tools, users must explicitly specify which objects to show at each zoom level of their map, while keeping their application constraints implicit. This paper introduces a novel declarative approach to map generalization based on a language called CVL, the Cartographic Visualization Language. In contrast to current tools, users declare application constraints and object importance in CVL, while leaving the selection of objects implicit. In order to compute an explicit selection of objects, CVL scripts are translated into an algorithmic search task. We show how this translation allows for reuse of existing algorithms from the optimization literature, while at the same time supporting fully pluggable, user-defined constraints and object weight functions. In addition, we show how to evaluate CVL entirely inside a relational database. The latter allows users to seamlessly integrate storage of geospatial data with its transformation into map visualizations. In a set of experiments with a variety of real-world data sets, we find that CVL produces generalizations in reasonable time for off-line processing; furthermore, the quality of the generalizations is high with respect to the chosen objective function.**

I. Introduction

The goal of map generalization is to produce a map at a given scale that achieves the right balance between rendering performance and information quality for end users. For example, in a tourist attraction rating system, one needs to efficiently visualize important attractions, and constrain object proximity to allow space for user interaction. In a journalistic piece that maps traffic incidents, however, maintaining the underlying distribution of data is the most important aspect, but at the same time object density must be constrained to ensure high-performance data transfer and rendering.

Fully automatic generalization of digital maps [1], [2] is relevant in many areas such as social networks, factivism and data journalism [3], [4], [5], where there is a constant need for visualizing new and often massive geospatial datasets. Automatic generalization includes both data reduction and graphical rendering [6], [7]. Increasingly, graphical rendering is deferred to map clients. This trend leaves the challenging problem of data reduction, i.e., selecting the right information to be displayed across zoom levels of the map, to the map service provider [8].

Both the performance and quality of a generalized map become important as the map gains a large audience. A map generalization solution handling data reduction in this context should be able to deal with big spatial datasets, consisting of both point and polygon records, should be usable by novice programmers, and should be able to finish processing quickly, e.g., in time for a tight news agency deadline. Ideally, such a system will allow users to control the important aspects of generalization solutions using logical and concise measures and reuse existing technology as much as possible, e.g., relational database technology.

Unfortunately, current approaches for data reduction in map generalization fall short in one or many of the above dimensions. Recent work has mostly considered only explicit rules or pre-set constraints for map generalization, resulting in solutions that are either too tedious [9], [10], or too restrictive for users [1], [2]. In addition, previous solutions have been poorly integrated with existing technology, resulting in scalability bottlenecks such as being restricted to the main memory capacity of a single node [1].

Spatial data is often stored in a database with powerful spatial extensions installed, so a natural idea is to exploit the processing capabilities of the database to perform map generalization. In this work, we present a novel *database-integrated* approach that is a complete solution to the data reduction problem in map generalization. All operations are performed entirely within the database process, and the result is a preprocessing of spatial records for fast execution of subsequent scale-parameterized queries [11]. Essentially, a number is assigned to each spatial record corresponding to the lowest zoom-level at which the record should be visible in a zoomable map, allowing for efficient indexing.

Using a *declarative language*, we allow the user to concisely express spatial constraints and object importance, which are used to compute a multi-scale database from an input table of spatial data. This gives users a large amount of control over the map generalization process, while still being extremely concise, expressing a generalization with as little as four lines of code.

We term our approach *declarative cartography*, since it combines a declarative language for data reduction with a compilation procedure that results in efficient database programs to transform data for cartographic visualization.

In this paper, we make the following four contributions:

1) We present a declarative language, Cartographic Visualization Language (CVL, pronounced "civil"), for generalizing spatial datasets. CVL is designed to be simple and concise to use for novice programmers. The CVL language was designed in collaboration with the Danish Geodata Agency and Grontmij in Denmark.[1,2]
2) We convert the data reduction problem in map generalization to an instance of the well-known *set multicover problem* [12], which makes constraints fully pluggable and allows reuse of well-known algorithms [12], [13].
3) We show how to fully evaluate CVL inside the database; this enables us to reuse basic database technology for data management and scalability. While CVL is designed to compile to a variety of engines [14], we present here an implementation using a relational database engine with spatial extensions. The code for the project is available as open source through the project website.[3]
4) We present experimental results for a variety of real datasets. The results show that the proposed approach has good performance and produces high-quality map generalizations.

In Section II, we define the data reduction problem in map generalization as a selection problem. In Section III, we introduce the CVL language. In Section IV, we formalize the selection problem as a combinatorial optimization problem based on a mapping to the set multicover problem, and we revisit algorithms for this problem in Section V. In Section VI, we discuss the compilation procedure that enables us to run CVL on a relational database backend. Experimental results are presented in Section VII, and finally related work is summarized in Section VIII.

II. Selection of Geospatial Data

In the *selection problem,* we wish to select the subset of a geospatial dataset to be visualized on a map at a given scale. Below we define the basic components of the problem, and informally define the associated optimization problem.

A. Geospatial records and weights

The dataset is assumed to consist of a set of *geospatial records* drawn from a database table. The schema of a geospatial record consists of a *geometry* field (e.g. a point, line or polygon), a *unique ID* field and any number of additional textual and numeric fields, such as "city name" and "population".

Each record is assigned a *user defined weight* using CVL (see Section III). The weight models the importance of a record, with high weight corresponding to great importance. Any subset of records — or all records for that matter — may have the same weight. Therefore, the weights induce a partial order of the records.

[1]http://www.gst.dk/English/
[2]http://grontmij.dk/
[3]http://github.com/dmslab/declarativecartography

B. Zoom levels and map constraints

For zoomable maps, different subsets of the data should be selected for display at different scales or *zoom levels*. Let the zoom-levels run from 1 (lowest scale) to $\mathcal{Z}$ (largest scale). On a given zoom level, the map is rendered at a certain pixel resolution. Thus, for a given zoom level, we know the distance in pixels between geospatial locations. This gives rise to two particularly important map constraints [15] when selecting data for a given zoom level.

Firstly, the *principle of constant information density* implies that the number of records that can be displayed within an area of a certain pixel size should be bounded [16]. Assume that we divide the complete map into cells (or tiles) of, say, 256 x 256 pixels. The *visibility* constraint states that each cell can contain at most K selected records, where K is a user-defined parameter [1].

Secondly, records cannot be too close to each other in the map — otherwise the user will not be able to clearly distinguish between them. The *proximity* constraint states that every pair of visible records must be separated by at least d pixels, where d is a user defined parameter.

In addition to these constraints that must hold separately for each zoom level, there are constraints that must hold across zoom levels. A particularly important constraint is the *zoom-consistency* constraint, which states that when a record is filtered out at a given scale, it should also be filtered out at all *lower* scales [1]. When a user zooms out on a map, records can only disappear — not reappear.

Apart from the zoom-consistency constraint, CVL supports constraints based on *simple measures* that are *satisfiable by selection* (see Section III). A simple measure is a function that maps a set of records to a scalar value. A constraint is violated if the measure exceeds a threshold. A constraint is satisfiable by selection if we can *always* satisfy it by simply deleting an appropriate subset of the records. Both the visibility and proximity constraints respect these restrictions. However, we cannot model constraints that have complex measures or cannot be satisfied by using selection alone, such as *topology* and *spatial distribution* constraints. We leave these classes of constraints to future work.

C. Conflicts

Constraints such as visibility or proximity can be modeled using the notion of *conflicts*. A conflict is a set of records that cannot all be selected without violating the constraint.

For the visibility constraint, there is a conflict generated for every cell that contains more than K records. For the proximity constraint, there is a conflict generated for each pair of records that is less than d pixels apart (see Figure 1). A record can be in several conflicts, which is the case for point p in the example shown in the figure. A solution to the selection problem is *feasible* if there are no conflicts.

Consider a conflict involving k_1 records, where at most k_2 of these records can be selected (where $k_1 > k_2$). Then it is equivalent to state that at least $\lambda = k_1 - k_2$ of these records must be *deleted*. In the mathematical formulation of

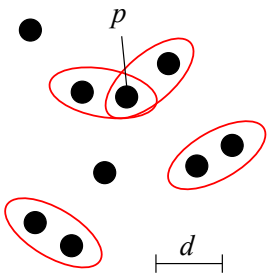


Fig. 1. Conflicts generated by the proximity constraint for distance d. Notice that point p is a member of more than one conflict.

the problem in Section IV, we will use this alternative way to formulate conflicts.

D. Selection as an optimization problem

The notion of conflicts is used to define the feasibility of solutions to the selection problem. This should be accompanied by a way to discriminate between solutions. Assigning an importance measure to each record, namely the record weights, intuitively allows us to measure the "loss of importance" due to records that are deleted.

In the optimization version of the problem, we seek the feasible solution that minimizes the aggregate weight of records that are deleted. In Section IV, we present a mathematical formulation of the selection optimization problem.

For a zoomable map with $\mathcal{Z}$ zoom levels, we are interested in finding $\mathcal{Z}$ solutions to the selection problem, one for each zoom level $i \in \{1, \ldots, \mathcal{Z}\}$. We call this problem the *multi-scale selection problem*. To control the way in which we compute these solutions, we use an algorithmic framework known as the *ladder* approach [17]. This is a recursive approach, where the output of selection at large scale is used as input to selection at a smaller scale. This means that the zoom-consistency constraint (Section II-B) is automatically satisfied.

The ladder approach is not appropriate for all use cases. For example, when regional labels are modeled as geospatial records, e.g., the label "Europe", we may wish to show a record only on intermediate zoom levels, violating zoom consistency. Handling these use cases would require an alternative formulation, e.g., following the *star* approach [17]. This is an interesting avenue for future work.

III. CVL Language

The Cartographic Visualization Language (CVL) is a declarative language that can be used to specify an instance of the multi-scale selection problem (Section II-D). CVL is a rule-based language with a similar goal as other rule-based languages for selection over spatial datasets, i.e., to control the density of information at each zoom-level [9], [10]. The CVL approach is, however, markedly different. In the related languages, the user must explicitly control the selection of records at each zoom level, while also specifying how records are to be visualized. First of all, CVL focuses only on selection, not presentation. Furthermore, CVL controls selection in a novel constraint-based way. Instead of having the user explicitly control the selection of records at each zoom level, CVL lets the user choose *map constraints* that are instead enforced at all zoom levels. By making the constraints explicit and the control implicit, a very concise formulation is obtained (see Figure 4 for an example).

```
GENERALIZE
   {input} TO {output}
WITH ID {expression}
WITH GEOMETRY {expression}
AT {integer} ZOOM LEVELS
WEIGH BY
  {float expression}
SUBJECT TO
   {constraint} {float parameters} [AND
   {constraint} {float parameters} [AND
   ...]]
```

Fig. 2. Syntax of generalize statement.

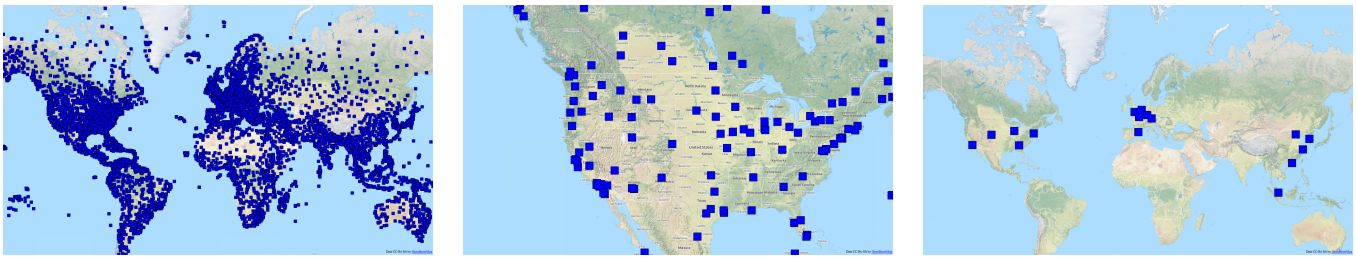
CVL is one of the first languages and frameworks to implement the vision of reverse data management [18]. In reverse data management, the core idea is that a user states a set of constraints and an objective. These are given together with an input database to an optimization algorithm which computes an output database that is feasible and optimal with regard to the constraints and objective (if a feasible solution exists). This is exactly how CVL works. Furthermore, a feasible solution is guaranteed to exist, as deleting all records is always a feasible solution.

The CVL language has two statements, the *generalize* statement (see Section III-A) and the *create-constraint* statement (see Section III-B). The create constraint statement is used to formulate new map constraints and the generalize statement is used to specify a solution to the multi-scale selection problem subject to those constraints.

The CVL language builds on top of SQL and reuses SQL as a language for formulating constraints and record weighting schemes.

A. Generalize statement

The generalize statement is the main statement in CVL. This statement creates a new multi-scale dataset from an input table of geospatial records, subject to user defined constraints. The syntax is shown in Figure 2. The statement has several clauses, beginning with the specification of input and output tables. Instead of giving the name of an input table, the user can optionally write a select statement in SQL of the form `(SELECT ...) t`. The next clause is the *with-id* clause, which is used to uniquely identify records. If records have an id column, this clause could simply provide the name of that column. The *with-geometry* clause is used to indicate the geometry property of records, e.g., the name of an available geometry column. The next clause is the *zoom-levels* clause where the user writes a positive integer, which is the highest zoom level at which the selection process will begin. The *weigh-by* clause is used to give an arbitrary floating point expression that is evaluated for each row in the input and used as weight for that record. The *subject-to* clause lists the map



(a) Full Openflights Airport dataset (b) Airports on zoom-level 5 (c) Airports on the top-most zoom-level.

Fig. 3. Airport map (7K points) before (a) and after (b, c) running CVL. The output corresponds to the CVL statement in Figure 4.

constraints along with any parameters (as a comma-separated list). The `AND` keyword is used to separate constraints in the case more than one is used.

An example of generalizing a dataset using the generalize statement is shown in Figures 3 and 4. In this example a dataset containing point records representing the location of airports world-wide is generalized (Figure 3(a)). The records are weighted by using the name of a column containing the number of routes departing from each airport (shown in Figure 4; CVL automatically handles the cast from integer to floating point). The intuition is that airports with more departures are more important. The single constraint that is enforced is the visibility constraint, with a parameter of $K = 16$. Recall that the visibility constraint says that each tile can contain at most K records.

```
GENERALIZE
   airports TO airports2
WITH ID airport_id
WITH GEOMETRY wkb_geometry
AT 18 ZOOM LEVELS
WEIGH BY
  num_departures
SUBJECT TO
   visibility 16
```

Fig. 4. Generalization of an airports dataset. The airports are weighted by number of departures. See Figure 3 for a vizualization of the result.

The resulting map is shown in Figure 3(b) and (c) and has at most 16 airports on each tile. For the single tile on the top zoom-level, the world's sixteen busiest airports are shown. The CVL framework automatically gives priority to the airports with the highest weight. How this is done is explained in sections IV and V.

B. Create constraint statement

Map constraints are defined using the create-constraint statement. The basic syntax of the statement is shown in Figure 5. The body of the statement is a SQL select statement that computes tuples that represent conflicts that are found at a given zoom level in the map. A tuple $\langle cid, rid \rangle$ denotes that record rid is a member of conflict cid. See Section II-C for the exact semantics of conflicts.

The *resolve-if-delete* clause is used to compute the integer number of records that must be deleted in order to resolve the conflict with a given cid.

```
CREATE CONSTRAINT C1
AS NOT EXISTS
  {SQL select statement}

RESOLVE cid IF DELETE (
  {integer expression}
)
```

Fig. 5. Syntax of create constraint statement

```
CREATE CONSTRAINT Proximity
AS NOT EXISTS (
  SELECT
    l.{rid} || r.{rid} AS cid,
    Unnest(array[l.{rid}, r.{rid}]) AS rid
  FROM
    {level_view} l
  JOIN
    {level_view} r
  ON
    l.{rid} < r.{rid}
  AND
    l.{geom} && ST_Expand(r.{geom},
      CVL_Resolution({z}, 256) *
        {parameter_1})
  AND
    ST_Distance(l.{geom}, r.{geom}) <
      CVL_Resolution({z}, 256) * {parameter_1}
)

RESOLVE cid IF DELETE (
  1
)
```

Fig. 6. Definition of the proximity constraint.

Using this syntax, the definition of the proximity constraint is given in Figure 6. The body of the constraint is a distance self join using a distance function `ST_Distance` provided by a spatial extension to SQL. This join finds all pairs of records that are too close, e.g. less than 10 pixels apart. For each conflict, the select statement outputs two tuples and exactly once for each conflict. The resolve-if-delete clause is simply the constant 1, because that is how many records must be deleted to resolve a proximity conflict.

In Figure 6, some names are enclosed in curly braces, such as `{rid}`. These are variables which are bound at runtime by

the CVL framework and are intended for making the definition of constraints simpler. The variables {rid} and {geom} are bound to the column names containing the ID and geometry of the records. The {level_view} is bound to a view that contains all records that are visible at the current level, i.e., the records that have not been filtered out at a higher zoom-level. The function CVL_Resolution({z}, 256) is one of the utility functions defined by the CVL runtime, also with the purpose of making the definition of constraints simpler. This function returns the resolution (meter/pixel) at zoom-level {z}, where {z} is a variable bound to the currently evaluated zoom-level. The variable {parameter_1} is the constraint parameter, e.g. 10 pixels.

```
CREATE CONSTRAINT Visibility
AS NOT EXISTS (
    SELECT
        busted_tiles.cid,
        busted_tiles.rid
    FROM
        busted_tiles
)

RESOLVE cid IF DELETE (
  SELECT count(*) - {parameter_1}
  FROM   busted_tiles bt
  WHERE  bt.cid = cid
)

WITH SETUP (
    CREATE TEMPORARY TABLE busted_tiles AS (
        SELECT
            t.cid,
            Unnest(array_agg(t.cvl_id)) AS rid
        FROM
        (
        SELECT
            CVL_PointHash(CVL_WebMercatorCells
                ({geometry}, {z})) AS cid,
            {rid}
        FROM
            {level_view}
        ) t
        GROUP BY t.cid
        HAVING count(*) > {parameter_1}
    );
    CREATE INDEX busted_tiles_id_idx ON
        busted_tiles (cid);
)

WITH TEARDOWN (
  DROP TABLE busted_tiles;
)
```

Fig. 7. Definition of the visibility constraint.

Figure 7 shows how the visibility constraint may be defined using CVL. The CVL definition uses an extension of the basic create-constraint syntax, namely the *setup* and *tear down* clauses. The purpose of these clauses is to enable arbitrary SQL statements to be run before and after the constraint body is evaluated at each zoom-level. During the setup phase we create an auxiliary table called busted_tiles which contains tuples $\langle tile\_id, rid \rangle$ identifying tiles that are intersected by more than K records, and the ID of those records. The body of the constraint simply iterates over the auxiliary table, using the tile_id column as the conflict ID.

The user does not need to know how the conflicts are handled, because all conflicts are automatically resolved by the CVL framework using one of the algorithms presented in Section V.

IV. Selection optimization problem

In this section, we formally define the selection problem as an optimization problem. Let R be the set of records in the dataset. Each record $r \in R$ has an associated weight $w_r > 0$ which models the importance of the record.

Evaluating a CVL query generates a number of conflicts, i.e., all sets of records that violate a constraint. Let C be the set of conflicts. A conflict $c \in C$ is a set of records $R_c \subseteq R$, where at least $\lambda_c \geq 1$ records must be deleted. The selection problem can now be modeled as a 0-1 integer program. Let x_r be a 0-1 decision variable for each record $r \in R$ that is 1 if record r is *deleted*, and 0 otherwise. Then at a single-scale, the problem can be stated as follows:

$$\min \sum_{r \in R} w_r x_r \tag{1}$$

$$\sum_{r \in R_c} x_r \geq \lambda_c, \qquad c \in C \tag{2}$$

$$x_r \in \{0, 1\}, \quad r \in R \tag{3}$$

The goal (1) is to minimize the total weight of the records that are deleted. The inequalities (2) model the conflicts in the selection optimization problem. This is the *set multicover problem* — a generalization of the well-known set cover problem where each element needs to be covered multiple times instead of just once [12]. In our formulation, conflicts correspond to *elements* in the set multicover problem, while records correspond to *sets*. Each conflict $c \in C$ must be "covered" $\lambda_c \geq 1$ times by choosing a subset of records that are deleted (i.e., for which $x_r = 1$). Because the selection of records is modeled using a 0-1 decision variable, each record can be chosen at most once.

The general selection optimization problem is clearly equivalent to the set multicover problem. Since the set multicover problem is NP-hard, so is the general selection optimization problem. The selection optimization problem is even NP-hard for very restricted cases. Consider the vertex cover problem: Given a graph $G = (V, E)$, find a minimum size subset of the vertices S such that every edge in E has an endpoint in S. The vertex cover problem is equivalent to the restricted case of the selection optimization problem where all records have unit weight, and all constraints contain exactly two records. (The records are the vertices and the conflicts are the edges in the vertex cover problem.)

The vertex cover problem is NP-hard, even for very restricted cases. For example, if G is a planar graph and every

vertex has degree at most 3, the problem remains NP-hard [19], [20]. This corresponds to a selection optimization problem where the conflicts contain two records each, and each record is involved in at most 3 conflicts. It is hard to imagine that any interesting application is more restrictive.

In the next section, we discuss algorithmic approaches for solving the selection optimization problem. We include a further discussion on the objective value (1) in the experimental evaluation of our approach.

V. Algorithms for selection problem

As mentioned in Section II-D, we solve the multi-scale selection problem using the ladder approach. For each of the $\mathcal{Z}$ zoom levels, we generate and solve a separate instance of the selection optimization problem (Section IV). The solution gives us the records that should be deleted from zoom-level i. The remaining records are (conceptually) copied to zoom level $i-1$, unless we have reached the last zoom level ($i = 1$). This approach is illustrated schematically in Figure 8.

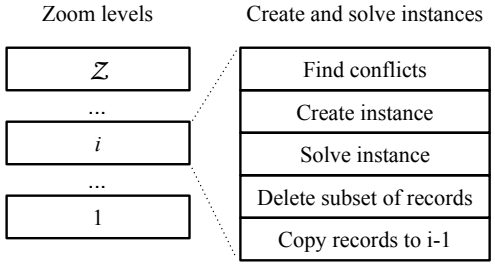


Fig. 8. Algorithmic framework: At each zoom level $i \in \{1, \ldots, \mathcal{Z}\}$ we solve a selection optimization problem. In the ladder approach, the problem is solved for the "highest" zoom level first.

Below we describe two different heuristic algorithms for solving the selection optimization problem. Let $n = |C|$ be the number of conflicts (or elements in the set multicover problem), and let $m = |R|$ be the number of records (or sets in the set multicover problem). Recall that $R_c \subseteq R$ is the set of records in conflict $c \in C$. The largest number of records in any conflict is $f = \max_{c \in C} |R_c|$, and is called the *maximum frequency*.

A. Static greedy algorithm (SGA)

In this algorithm, we consider each conflict $c \in C$ in turn, and simply choose the λ_c records with minimum weight from the records R_c — independently of what has been chosen earlier. If the sets R_c are disjoint, the algorithm is clearly optimal. However, in general no approximation guarantee can be provided. The algorithm runs in $O(nf \log f)$ time, as we just need to sort the records by weight for each conflict set; alternatively we can sort all records by weight in $O(m \log m)$ time and pick the minimum weight records from the conflicts in linear time in the total number of records in all conflict sets.

B. LP-based greedy algorithm (LPGA)

In this algorithm, we first solve a linear programming (LP) relaxation of the set multicover problem. This LP-problem is obtained by relaxing the constraint $x_r \in \{0, 1\}$ to $0 \leq x_r \leq 1$. Then we choose all records $r \in R$ for which the LP-solution variable x_r is at least $1/f$. Intuitively, we round up to 1 all fractional values that are large enough; the remaining fractional variables are rounded down to 0.

This algorithm provides a feasible solution to the selection optimization problem, and the approximation guarantee is f [13]; thus, if f is small, the algorithm provides a good approximation guarantee. As the LP-problem can be solved in polynomial time, the complete algorithm is polynomial.

VI. Implementation

In this section, we describe how our implementation makes use of in-database execution to provide scalability and engine reuse for CVL (Section VI-A). In addition, we discuss a number of extensions to CVL that we found to be useful for practical applications (Section VI-B).

A. In-Database Execution

Overview. Since CVL is declarative, and CVL constraints are already expressed in SQL, it is natural to attempt to reuse as much existing DBMS technology as possible to execute CVL. Figure 9 shows how CVL is compiled for execution in a relational DBMS, which acts as the language runtime. The output of the CVL compiler is a database script for the target host, containing both SQL and stored procedures, and following the algorithmic framework of Figure 8. The script is pushed down to the database engine, and operates against the appropriate input data stored in the system. This strategy offers us two main advantages:

1) Since all code is pushed down and both input and output reside in the database, we do not need to transfer any data outside of the database engine. This co-location of code and data is a significant advantage for large datasets.
2) By expressing as much as possible of the generated code in SQL, we can reuse decades of optimizations built into database engines, especially for geospatial data [21], [22]. This opens up many opportunities, such as automatic optimization, parallelism, and selection of specialized algorithms and indexes.

While the general strategy of compiling declarative languages to SQL has been pursued in other contexts, e.g., for XQuery [23] and LINQ [24], our context poses a particular challenge of integrating the language with algorithmic solvers inside the database.

Solvers. In Section V, we presented two different algorithmic approaches for solving CVL generalizations: static greedy (SGA) and LP-based greedy (LPGA). We now show how to express each of these approaches in SQL along with stored procedures.

SGA is the simplest algorithm, and operates independently on the conflicts generated by each constraint. Suppose the

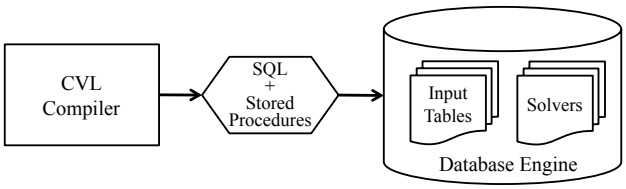


Fig. 9. CVL and in-database execution.

conflicts C generated by the active constraints are stored in a *conflicts* table. Then SGA is tantamount to the query:

```
SELECT rid
FROM (
  SELECT ROW_NUMBER()
         OVER (PARTITION BY cid
               ORDER BY cvl_rank) AS r,
         rid, cvl_rank, lambda_c
  FROM conflicts) h
WHERE h.r <= h.lambda_c
```

For each conflict c, we order records by rank, and ensure that we pick at least λ_c records. The declarative formulation allows us to reuse optimized sorting code in the database engine for execution.

LPGA solves a linear programming relaxation of the set multicover problem. We express LPGA by a stored procedure. The procedure accesses the conflicts for the constraints via SQL, constructs an appropriate LP, and then calls into an LP solver library. Since the solver library does not use built-in database optimizations, this execution strategy for LPGA only leverages the first advantage of data and code co-location listed above.

Finally, note that the code for finding conflicts is already expressed in SQL by the user for each constraint. As a consequence, this user code can make use of all built-in database optimizations available in the target engine.

CVL runtime functions. In the definition of the visibility constraint in Section III-B, we reference two stored procedures in the CVL runtime library, `CVL_PointHash` and `CVL_WebMercatorCells`. These functions are implemented in SQL and make use of the spatial extension of the database.

The procedure `CVL_PointHash` uses a call to `ST_GeoHash` to implement an injective mapping from points to strings. The GeoHash algorithm corresponds to a Z-order curve, and we exploit this for uniquely naming tiles when evaluating the visibility constraint, i.e. finding tiles with more than K records.

The `CVL_WebMercatorCells` function maps a geometry at a given zoom level to centroids of all intersected tiles (on that zoom level). We experimented with several ways to do this for general geometries (points, line segments, polygons) and found that rasterizing the geometry (using the function `ST_AsRaster` in the spatial extension of the database) and iterating over the indices was the fastest for general geometries. For point records it is significantly faster to use the standard transformation function `ST_SnapToGrid`.

B. Extensions

When designing CVL, we realized a number of interesting use cases for the language that we had not initially considered. This realization, along with our implementation experience of CVL use cases, led us to a set of extensions over the core language targeted at improving convenience of use. We present these extensions below.

Partitioning and merging of datasets. A single input table may contain geospatial objects of different classes, e.g., roads and points of interest. When this is the case, users often wish to generalize some of these classes of objects independently, but obtain a single result map. While this can be done by merging the results of multiple GENERALIZE statements, we found it useful to add syntactic sugar to support this case. We extend the GENERALIZE statement with PARTITION BY and MERGE PARTITIONS clauses. PARTITION BY allows us to effectively segregate the input into multiple independent sets. MERGE PARTITIONS combines a few of these sets back together before providing them as input to generalization. For example, assume a *geo_objects* table contains highways, roads, restaurants, and hotels, tagged by a *type* attribute. We could then generalize *geo_objects* as follows:

```
GENERALIZE  geo_objects
TO network_and_poi_map
...
PARTITION BY type
MERGE PARTITIONS 'restaurant', 'hotel'
              AS 'poi'
...
```

In the example, we overlay independent generalizations of highways, roads, and points of interest into a single map. However, restaurants and hotels are generalized as a single input set.

Forced and all-or-nothing visualization. Intuitively, constraints let users specify what is *not* allowed in a given map, by forbidding the existence of conflicts. However, users also find it helpful to control certain behaviors that *must* occur in their map. We extended the GENERALIZE statement with support for two types of behaviors: (1) the ability to mandate a minimum zoom level for a particular partition of the input, and (2) the ability to force that either all or none of the objects of a given partition be displayed. For example, a user may wish to specify that highways must only appear at zoom level 10 or lower in their map. In addition, for topological consistency, either the whole highway skeleton is displayed or no highways should show up. To achieve this goal, we extend the GENERALIZE statement by a FORCE clause with MIN LEVEL and ALLORNOTHING specifiers. Continuing the example above:

```
...
FORCE MIN LEVEL 10 FOR 'highway' AND
ALLORNOTHING FOR 'roads'
...
```

In the evaluation of CVL, the minimum level specifier controls what data is given as input for a zoom level. The all-or-nothing specifier, on the other hand, controls filtering of the output of the level generalization process. If the specifier

is present, all records of a partition are deleted if any record from the partition input is not present in the output. By filtering output, we ensure that the result also respects all other constraints specified by the user.

VII. Experimental Results

In this section, we present experimental results with our implementation of CVL. Our experiments have the following goals:

- Evaluate the performance and solution quality of CVL generalizations with a variety of real-world datasets, including point data as well as complex shapes such as polygon and line data.
- Analyze the performance and solution quality of CVL generalizations produced under the proximity and visibility constraints presented in Section III by both the SGA as well as the LPGA solvers of Section V.
- Observe how the performance of CVL with different constraints and solvers scales with the number of objects in the geospatial dataset.

We start by presenting our experimental setup (Section VII-A), and then show results for both point data (Section VII-B) and complex shapes (Section VII-C). Each result section discusses performance, quality, and scalability.

A. Experimental Setup

Datasets. We have tested CVL using four real-world datasets, the largest of which containing 9 million points, and one synthetic dataset containing 30 million points. We list all datasets in Table I.

We have used three point datasets. The airports dataset is from Openflights and contains 7411 airports.[4] The tourism dataset contains 500 thousand points representing tourist attractions worldwide from the OpenStreetMap database.[5] The fractal dataset (synthetic) was created by iteratively copying and displacing points from the tourism dataset within a 10km radius until 30 million records were reached. We use this dataset for scalability experiments.

We have used two line datasets. The US rivers/streams dataset contains roughly 4 thousand rivers and roughly 27 thousand streams in the United States from the OpenStreetMap database. Records with identical name attributes have been merged into one. In the original dataset, most rivers are represented by multiple records, which is unfortunate in a selection situation (we wish to either select the waterway completely or not at all).

We have used a single polygon dataset, the area information dataset from The Danish Natural Environment Portal, published by the Danish government.[6] This dataset contains 30 thousand high-fidelity administrative protection zone polygons, ranging from small polygons the size of buildings to large polygons the size of entire regions. The largest polygon has more than 36 thousand vertices.

[4]http://openflights.org/data.html

[5]http://www.openstreetmap.org/

[6]http://internet.miljoeportal.dk/

We have tested the scalability of CVL using both point and line datasets. A east-west unrolling approach is employed for gradually increasing the size of a dataset. First, we order records by x-coordinate, and then select increasingly larger prefixes of this order to derive larger datasets. The advantage of this approach over random sampling is that the spatial density of records is better preserved.

TABLE I
Datasets used in experiments

Origin	Dataset	Type	Records	Points
Real	Airports	Points	$7K$	$7K$
Real	Tourism	Points	$500K$	$500K$
Synthetic	Fractal	Points	$30M$	$30M$
Real	US rivers	Line segments	$4K$	$2M$
Real	US rivers/streams	Line segments	$30K$	$6M$
Real	Proctection zones	Polygons	$30K$	$9M$

Hardware, software, and methods. The machine used for testing was an Amazon EC2 instance with 17GB RAM, 2 x Intel(R) Xeon(R) CPU E5-2665 0 @ 2.40GHz and 20MB cache, running Amazon Linux 3.4.48-45.46.amzn1.x86_64.[7]

The database used for testing was PostgreSQL 9.2.4 with PostGIS 2.0 built against the libraries GDAL 1.9.2 and GEOS 3.3.8. For the LP solver, we integrated the database with the convex optimization library CVXOPT version 1.1.6.[8] We installed Python language bindings in the database against Python 2.6.8.

We ran each test three times on this installation, taking averages. We observed that measurements were very stable, with negligible difference in compute time between runs.

PostgreSQL always uses a single core to compute a transaction. Because the generalization process in CVL runs as a single long transaction, each job in CVL runs on a single core. A future direction would be to investigate parallel execution of CVL queries using a different language runtime such as a parallel database or a MapReduce environment.

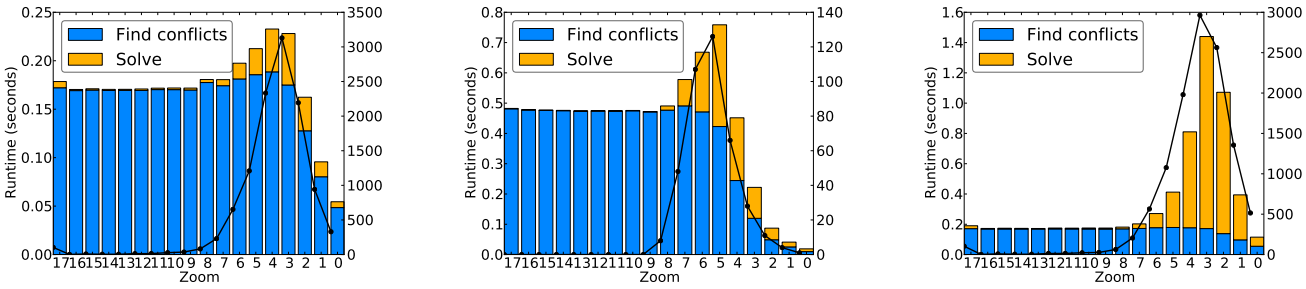
Average optimality ratio. In our approach, we solve the multi-scale selection problem as a series of selection optimization problems. To get an indication of the solution quality, we compute for every selection optimization problem a lower bound using an LP-relaxation of the integer program. The numbers we present in Table II and Table III include the average ratio between our solution value and the corresponding lower bound.

B. Point Data

In this section, we present experimental results with point datasets, namely the Openflight airports and the tourism datasets. We first discuss performance and quality for CVL and then proceed to analyze CVL's scalability behavior. Even though we experimented with all combinations of solvers

[7]An image of the instance we used for testing is available through Amazon EC2 as an AMI. More information is available on the website for the project.

[8]http://cvxopt.org/



(a) SGA + Proximity (b) LPGA + Visibility (c) LPGA + Proximity

Fig. 10. Performance breakdown by zoom level, Airport dataset (7K points). The black line indicates number of conflicts

(SGA / LPGA) and constraints (visibility / proximity / combined), we show only representative results for brevity. Results for the combined visibility and proximity constraints exhibited the same performance trends as of the most expensive of the two constraints. All other results followed similar trends as the ones explored below.

Overall performance and quality. An overview of running times and solution qualities for the point datasets are shown in Table II. In Section V-A, we remarked that SGA is optimal for disjoint conflict sets. This is confirmed by the entries for visibility + SGA in the table. For the point datasets we used for testing, the LPGA algorithm is also optimal or within 3% of the optimum when combined with the visibility constraint, likely caused by the conflict sets being disjoint. Recall that the approximation guarantee of LPGA is f (see Section V-B).

In terms of quality, the difference between SGA and LPGA is not stark for either constraint. The difference depends more on the constraint than on the solver, with visibility generally yielding the best solutions. However, the running time of SGA can be substantially shorter than that of LPGA. We analyze this effect in the following.

TABLE II
RESULTS FOR CVL ON POINT DATASETS GROUPED BY CONSTRAINT

Dataset	Constraint	Solver	Time	Avg. opt. ratio
Airports (7K)	Visibility	SGA	7s	1.0
Airports (7K)	Visibility	LPGA	7s	1.03
Tourism (500K)	Visibility	SGA	6m 9s	1.0
Tourism (500K)	Visibility	LPGA	13m 35s	1.0
Airports (7K)	Proximity	SGA	3s	1.18
Airports (7K)	Proximity	LPGA	7s s	1.22
Tourism (500K)	Proximity	SGA	7m 17s	1.21
Tourism (500K)	Proximity	LPGA	2h 18m	1.24

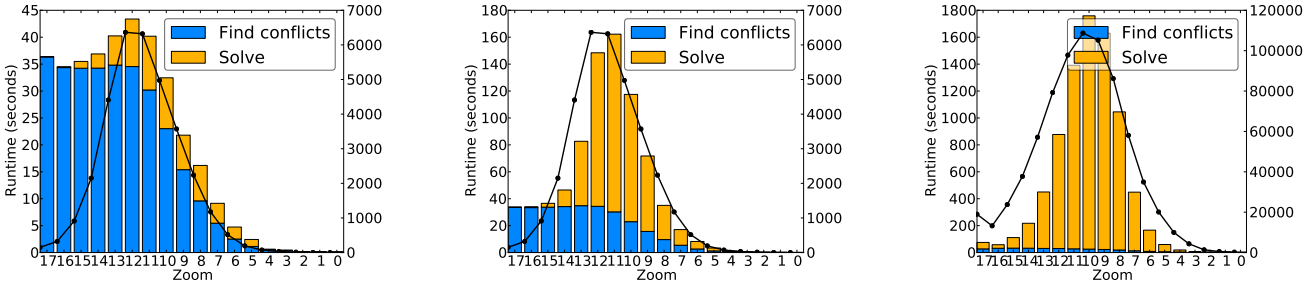
Performance breakdown. Figure 10 shows the performance breakdown per zoom level of executing CVL with the Openflight airports dataset. Note the different y-scales in the graphs. We have overlayed the number of conflicts per zoom-levels as a black line. In Parts (a)-(c), we observe that the time needed to find conflicts is roughly stable until eight zoom levels, then slightly increases, and finally drops sharply for lower zoom levels. The constraints used generate few conflicts at higher zoom levels, given the relatively low density of the airport distribution in space. Nevertheless, even though almost no conflicts are generated, the dataset is still processed, resulting in roughly equal time for finding conflicts and negligible time for solving conflicts per zoom level.

As zoom levels decrease, more conflicts naturally arise, leading initially to increased conflict finding time, as well as conflict solving time. However, as conflicts are solved, records are deleted from the dataset taken as input for the next zoom level. This procedure causes conflict finding time (and eventually total time) to drop significantly for low zoom levels. For SGA under the proximity constraint (Part (a)), total time at zoom level zero is over two times shorter than the initial runtime at zoom level 17; for LPGA under the visibility constraint (Part (b)), the difference in total time reaches over an order of magnitude.

Conflict solving time does not increase equally for different solvers. SGA exhibits conflict solving time that is consistently smaller than LPGA. Peak total time for SGA under the proximity constraint (Part (a)) is roughly four times shorter than for LPGA (Part (c)). In addition, LPGA is extremely sensitive to the number of conflicts reported by user-defined constraints. From Parts (b) and (c), we can see that LPGA exhibits peak conflict solving time over three times larger for the proximity constraint than for the visibility constraint, since the latter generates far fewer conflicts than the former.

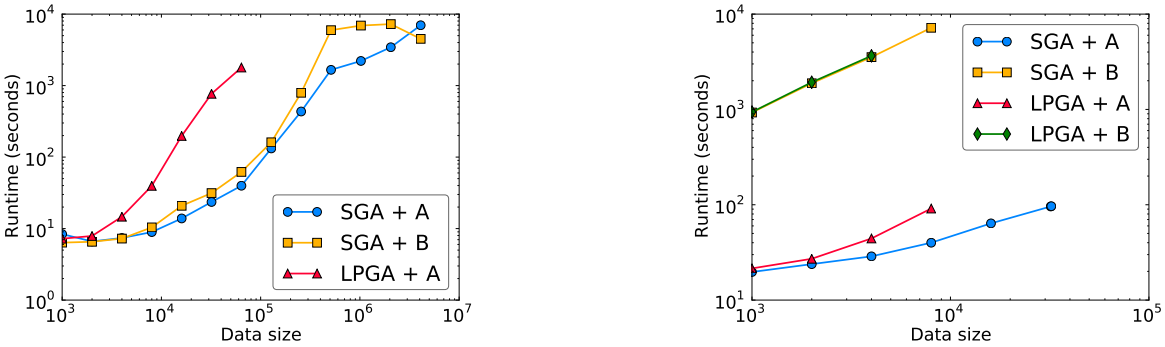
Figure 11 exhibits results with the larger tourism attraction dataset. Since the dataset is denser in space than the airport dataset, conflicts are found and solved at higher zoom levels, resulting in an earlier drop in total time per zoom level. For Parts (a)-(c), total time is uninteresting for zoom levels lower than five. The same cannot be said, however, about peak total time in general, and about conflict solving time in particular.

Parts (a) and (b) compare performance of SGA and LPGA under the visibility constraint. Even though visibility generates a smaller number of conflicts than proximity, peak total time for LPGA is still roughly a factor of four larger than for SGA (see zoom level 11). Note that the difference is completely due to the efficiency of the solver, since the time to find conflicts is essentially the same for both methods. Total time for LPGA rises prohibitively when we employ the proximity constraint, reaching a baffling peak of near half an hour at zoom level 10 (Part (c)). While not shown, total times per zoom level for SGA under the proximity constraint are roughly comparable



(a) SGA + Visibility (b) LPGA + Visibility (c) LPGA + Proximity

Fig. 11. Performance breakdown by zoom level, Tourism dataset (500K points). The black line indicates number of conflicts



(a) Scalability for point data (b) Scalability for complex shape data

Fig. 12. Scalability of CVL for point datasets and complex shape datasets. Constraints are marked as *Visibility*: A, *Proximity*: B

to the times reported in Part (a) for the visibility constraint using this dataset. SGA's peak total time is slightly above 40 seconds, roughly a factor of 40 smaller than LPGA's.

In summary, and as discussed in Section V-A, SGA performs significantly better than LPGA, but it does not do so at the cost of quality, at least for point datasets.

Scalability. We tested the scalability of CVL by varying the size of the synthetic dataset of 30 million points, starting with one thousand records, and tested by iteratively doubling up until we reached roughly four million records. We scaled the dataset with the sweep-line approach introduced in Section VII-A. We plot the running time of each solver/constraint combination for different dataset sizes in Figure 12.

In general, SGA scales far better than LPGA with the number of objects, confirming the observations from the performance breakdown above. After reaching four million points the running time became prohibitively large (more than 3 hours) even for SGA. Up to this point, the algorithm scales roughly linearly. The running time of the solvers depends on the number of conflicts, as well as on the structure of the conflicts. It is easy to see that after the first zoom-level, the number of conflicts is bounded by a constant that is proportional either to the number of records (for the proximity constraint) or the number of cells (for the visibility constraint). For the proximity constraint, the number of conflicts is bounded due to circle packing. For the visibility constraint, each cell can contain at most 64 records for $K = 16$, after the first zoom-level is processed. This is because each cell contains only records from four cells on the previous (higher) zoom-level, each of which contains only 16 records.

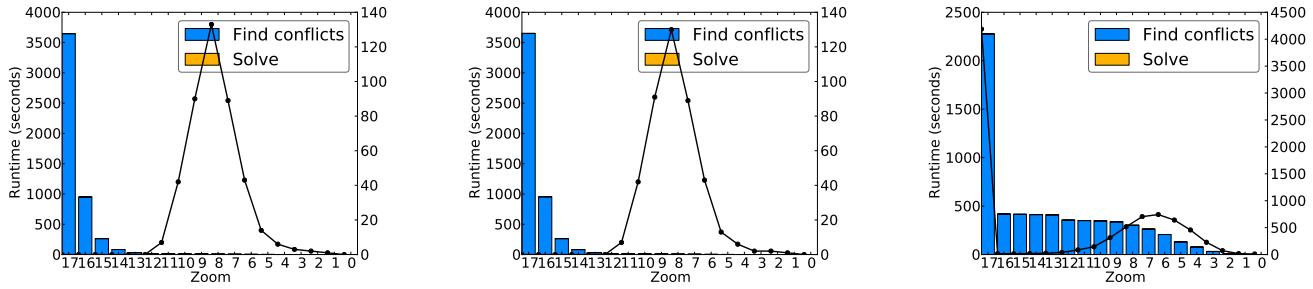
C. Complex Shape Data

Overall performance and quality. In Table III we summarize running times and average optimality ratios for complex shape data. We immediately observe that LPGA is now consistently better than SGA with regard to solution quality. This is in contrast to what we saw for points. We believe the cause to be that the conflict sets are no longer disjoint, and SGA suffers from this.

TABLE III
RESULTS FOR CVL ON COMPLEX DATASETS GROUPED BY CONSTRAINT

Dataset	Constraint	Solver	Time	Avg. opt. ratio
Rivers (4K)	Visibility	SGA	1h 32m	1.36
Rivers (4K)	Visibility	LPGA	1h 33m	1.0
Zones (30K)	Visibility	SGA	13m 38s	1.20
Zones (30K)	Visibility	LPGA	32m 15s	1.14
Rivers (4K)	Proximity	SGA	1h 11m s	1.46
Rivers (4K)	Proximity	LPGA	1h 31m	1.11
Zones (30K)	Proximity	SGA	4h 28m	1.72
Zones (30K)	Proximity	LPGA	—	—

Performance breakdown. In Figure 13, we show three performance breakdowns for the Rivers dataset. We make two observations. First, the running time is now completely dominated by finding conflicts. This is because the complexity



(a) LPGA + Visibility (b) SGA + Visibility (c) LPGA + Proximity

Fig. 13. Performance breakdown by zoom level, Rivers dataset (4K records). The black line indicates number of conflicts

of finding conflicts depends on the fidelity of the geometries that are compared. Parts (a)-(c) illustrate the effect in more detail, with Part (a) in particular showing the breakdown of a solution with an average optimality ratio of 1.0. We see that for complex shape datasets, the running time is mostly dominated by the time spent finding conflicts. Since finding conflicts operates over the geometric properties of the data, it requires time proportional at least to the number of points that make up each complex shape. When solving conflicts, the running time is independent of geometric complexity. Interestingly, the time necessary to find conflicts is so high that it shadows the negative effect that a larger number of conflicts has on the conflict resolution time of LPGA (compare with Section VII-B).

Scalability. In Figure 12(b), we show scalability results for complex shape data. Here scalability depends more on the choice of constraint than on the choice of solver. The proximity constraint scales much worse than the visibility constraint with the number of objects. This is because the running time of the distance test used in the proximity constraint is proportional to the product of point counts in the two geometric shapes used in each comparison. In contrast, evaluating the visibility constraint depends on the number of tiles that each shape intersects, which depends more on the length or area of each shape.

While constraints matter more to scalability for complex shapes than for point data, the SGA solver scales better than LPGA with number of objects, which was also the case for the point datasets examined in Section VII-B.

VIII. Related work

Cartographic generalization is a classic topic of investigation in the GIS community, and several models have been developed for generalization operations [15]. While the problem has been considered by some as AI complete [25], recent work has focused on automatic map generalization based on optimization models or queries for filtering [1], [2]. This reduction in scope reflects the need of providing a wide variety of web-accessible maps summarizing ever increasing amounts of geospatial datasets. Our work provides support for the same trend.

The optimization approach of Das Sarma et al. [1] is the most related to our work. In contrast to our approach, however, Das Sarma et al. do not provide a flexible declarative interface for user-defined constraints, nor does their approach leverage SQL. In addition, it is hard to integrate their approach with existing geospatial data serving infrastructures, which are mostly based on standard spatial database technology.

User-defined constraints and declarative specifications have been shown to yield elegant solutions to a variety of problems in data management, including record deduplication [26], database testing [27], [28], as well as cloud and networking resource optimization [29]. Our work brings these ideas to the context of map generalization and geospatial data, and as mentioned previously, is among the first frameworks to implement the vision of reverse data management [18].

In-database processing has also been explored successfully in diverse contexts in the literature. Translation of high-level languages, such as XQuery or LINQ, to SQL lead to highly scalable and efficient implementations [23], [24]. A number of recent approaches have targeted expressing complex statistical analytics in SQL [30], [31]. In contrast, we show how in-database processing can be used in the implementation of a declarative language for map generalization which includes solvers and constraints, leveraging the trend to incorporate whole programming language interpreters and support for spatial data structures in database engines [32].

Our approach dovetails with a number of techniques from the literature, which hold potential to further extend or complement it. First, we observe that the running time of the LP-based greedy algorithm (LPGA) is generally high. We implemented this algorithm because it provides a theoretical bound on the solution quality. We plan to explore other algorithms for set multicover, such as the greedy algorithm described by Rajagopalan and Vazirani [12], to improve running time compared to the LP-based greedy algorithm, while achieving good quality. An interesting challenge is how to express such algorithms entirely in SQL.

Second, this work considers only selection of objects. An important extension is to allow other data reduction operations, such as geometric transformation and aggregation of objects. While we believe that CVL could be adapted to these additional requirements, this would imply modeling alternative semantics and procedures for satisfying constraints in our framework.

Third, we would like to experiment with geospatially-aware parallel processing infrastructures, such as Hadoop-GIS [33], for even further scalability in computing map generalizations. Finally, once a map generalization is complete, the resulting map must be served to end-users. This problem is orthogonal to our work, and classic linearization techniques can be applied [11]. All of these are interesting avenues for future work.

IX. Conclusion

In this paper, we present a novel declarative approach to the data reduction problem in map generalization. The proposed approach integrates seamlessly with existing database technology, and allows users to specify, using the proposed CVL language, and across all scales, what goals and constraints the generated map should fulfill — leaving the detailed selection decisions to the system. The system leverages an algorithmic mapping which enables at the same time user-defined constraints and reuse of methods from the optimization literature. Our experiments show that the approach performs well for off-line processing and produces maps of high quality.

X. Acknowledgements

This project has been funded by Grontmij, the Danish Geodata Agency and the Ministry of Innovation in Denmark. We would like to thank employees and management at Grontmij and the Danish Geodata Agency for great discussions about the ideas presented in this work. We would also like to thank the anonymous reviewers for their insightful comments on our work. Finally, we would like to thank Amazon for providing an AWS in Education research grant award, which we used to implement our experimental setup.

References

[1] A. Das Sarma, H. Lee, H. Gonzalez, J. Madhavan, and A. Halevy, "Efficient spatial sampling of large geographical tables," in *Proc. 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. ACM, 2012, pp. 193–204.

[2] S. Nutanong, M. D. Adelfio, and H. Samet, "Multiresolution select-distinct queries on large geographic point sets," in *Proc. 20th International Conference on Advances in Geographic Information Systems*. ACM, 2012, pp. 159–168.

[3] S. Cohen, "Computational journalism: A call to arms to database researchers," in *Proc. Fifth Biennial Conference on Innovative Data Systems Research, CIDR 2011*, 2011, pp. 148–151.

[4] D. Gillmor, "'Factivism' for every field: why Bono is right to want more data and evidence," http://www.guardian.co.uk/commentisfree/2013/feb/28/bono-ted-talk-facktivist-is-way-forward, 2013, the Guardian. [Online; accessed 9-April-2013].

[5] J. Sankaranarayanan, H. Samet, B. E. Teitler, M. D. Lieberman, and J. Sperling, "Twitterstand: news in tweets," in *Proc. 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2009, pp. 42–51.

[6] R. Weibel and G. Dutton, "Generalising spatial data and dealing with multiple representations," *Geographical information systems*, vol. 1, pp. 125–155, 1999.

[7] D. Gruenreich, "Computer-Assisted Generalisation," *Papers CERCO Cartography course*, 1985.

[8] J. Gaffuri, "Toward Web Mapping with Vector Data," *Geographic Information Science*, pp. 87–101, 2012.

[9] O. G. Consortium, "Styled Layer Descriptor," http://www.opengeospatial.org/standards/sld, 2007, [Online; accessed 18-July-2013].

[10] A. Pavlenko, "Mapnik," http://mapnik.org/, 2011, [Online; accessed 18-July-2013].

[11] D. Hilbert, "Über die stetige Abbildung einer Linie auf ein Flächenstück," *Mathematische Annalen*, vol. 38, pp. 459–460, 1891.

[12] S. Rajagopalan and V. V. Vazirani, "Primal-dual RNC approximation algorithms for set cover and covering integer programs," *SIAM Journal on Computing*, vol. 28, no. 2, pp. 525–540, 1998.

[13] V. V. Vazirani, *Approximation algorithms*. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[14] M. Stonebraker, D. Abadi, D. J. DeWitt, S. Madden, E. Paulson, A. Pavlo, and A. Rasin, "Mapreduce and parallel dbmss: friends or foes?" *Commun. ACM*, vol. 53, no. 1, pp. 64–71, 2010.

[15] L. Harrie and R. Weibel, "Modelling the overall process of generalisation," *Generalisation of geographic information: cartographic modelling and applications*, pp. 67–88, 2007.

[16] F. Töpfer and W. Pillewizer, "The principles of selection," *Cartographic Journal, The*, vol. 3, no. 1, pp. 10–16, 1966.

[17] T. Foerster, J. Stoter, and M.-J. Kraak, "Challenges for automated generalisation at european mapping agencies: a qualitative and quantitative analysis," *Cartographic Journal, The*, vol. 47, no. 1, pp. 41–54, 2010.

[18] A. Meliou, W. Gatterbauer, and D. Suciu, "Reverse data management," *Proc. VLDB Endowment*, vol. 4, no. 12, 2011.

[19] P. Alimonti and V. Kann, "Some apx-completeness results for cubic graphs," *Theoretical Computer Science*, vol. 237, no. 1, pp. 123–134, 2000.

[20] M. R. Garey and D. S. Johnson, "The rectilinear steiner tree problem is np-complete," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[21] A. Guttman, "R-trees: a dynamic index structure for spatial searching," in *Proc. 1984 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '84. ACM, 1984, pp. 47–57.

[22] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer, "Generalized search trees for database systems," in *Proc. 21th International Conference on Very Large Data Bases*, ser. VLDB '95, 1995, pp. 562–573.

[23] P. Boncz, T. Grust, M. van Keulen, S. Manegold, J. Rittinger, and J. Teubner, "Pathfinder: XQuery — the relational way," in *Proc. 31st international conference on Very large data bases*, ser. VLDB '05. VLDB Endowment, 2005, pp. 1322–1325. [Online]. Available: http://dl.acm.org/citation.cfm?id=1083592.1083764

[24] T. Grust, M. Mayr, J. Rittinger, and T. Schreiber, "FERRY: database-supported program execution," in *Proc. 2009 ACM SIGMOD International Conference on Management of data*, ser. SIGMOD '09. New York, NY, USA: ACM, 2009, pp. 1063–1066. [Online]. Available: http://doi.acm.org/10.1145/1559845.1559982

[25] A. U. Frank and S. Timpf, "Multiple representations for cartographic objects in a multi-scale tree - an intelligent graphical zoom." *Computers and Graphics*, vol. 18, no. 6, pp. 823–829, 1994.

[26] A. Arasu, C. Re, and D. Suciu, "Large-scale deduplication with constraints using Dedupalog," in *Proc. 2009 International Conference on Data Engineering, ICDE'09*, 2009, pp. 952–963.

[27] C. Binnig, D. Kossmann, and E. Lo, "Reverse query processing," in *Proc. 2007 IEEE International Conference on Data Engineering, ICDE'07*, 2007, pp. 506–515.

[28] C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu, "QAGen: generating query-aware test databases," in *Proc. 2007 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '07. ACM, 2007, pp. 341–352.

[29] C. Liu, L. Ren, B. T. Loo, Y. Mao, and P. Basu, "Cologne: a declarative distributed constraint optimization platform," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 752–763, 2012.

[30] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar, "The MADlib analytics library: or MAD skills, the SQL," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 1700–1711, 2012.

[31] C. Ordonez, "Building statistical models and scoring with UDFs," in *Proc. 2007 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '07. ACM, 2007, pp. 1005–1016.

[32] J. A. Blakeley, V. Rao, I. Kunen, A. Prout, M. Henaire, and C. Kleinerman, ".NET database programmability and extensibility in Microsoft SQL server," in *Proc. 2008 ACM SIGMOD international conference on Management of data*, ser. SIGMOD '08. ACM, 2008, pp. 1087–1098.

[33] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz, "Hadoop-GIS: A spatial data warehousing system over MapReduce," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1009–1020, 2013.