

UNIVERSITY OF  
COPENHAGEN



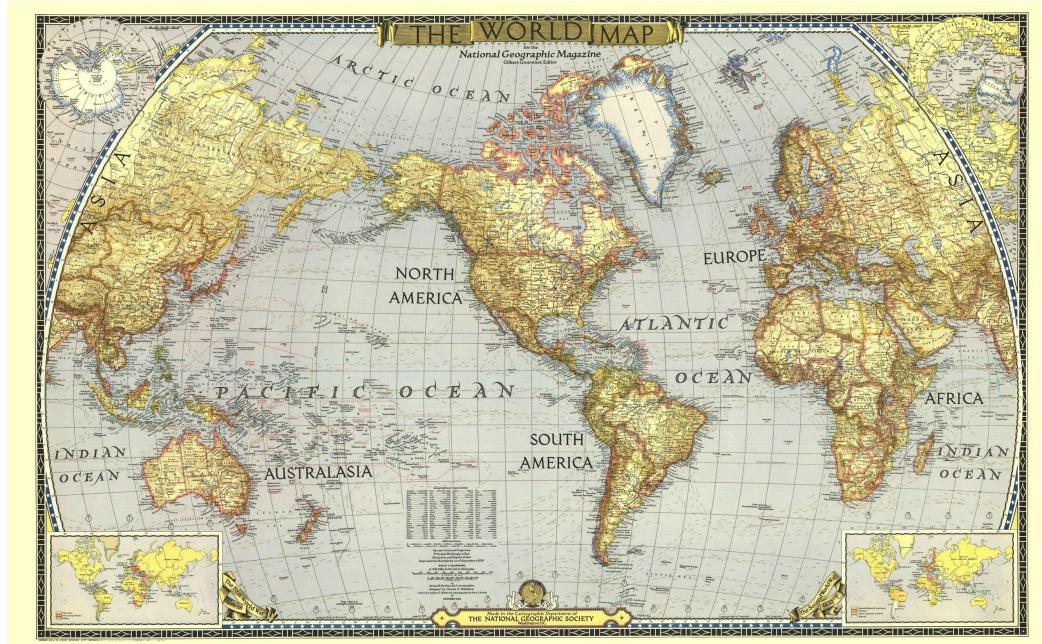
# Declarative Cartography

In-Database Map Generalization of Spatial Datasets

P.K. Kefaloukos, M. Vaz Salles, M. Zachariasen

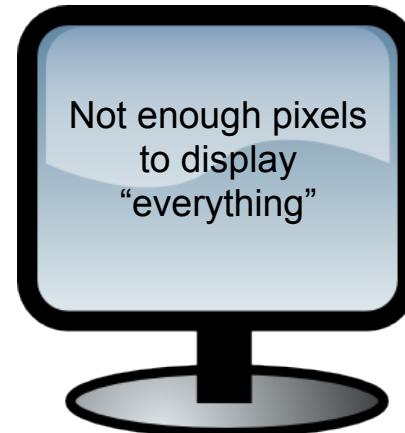
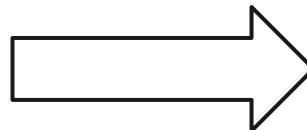
# Creating zoomable maps

- Maps should be
- Legible
  - Representative
  - Useful
  - Zoomable



# Basic challenge

Adapting data to scale of visualization medium



Not enough pixels  
to display  
“everything”

# Basic challenge

We have to *choose*:



(1) *What* to display?



(2) *How* to display it?

# Alternatives

## Selection



## Aggregation



# Alternatives

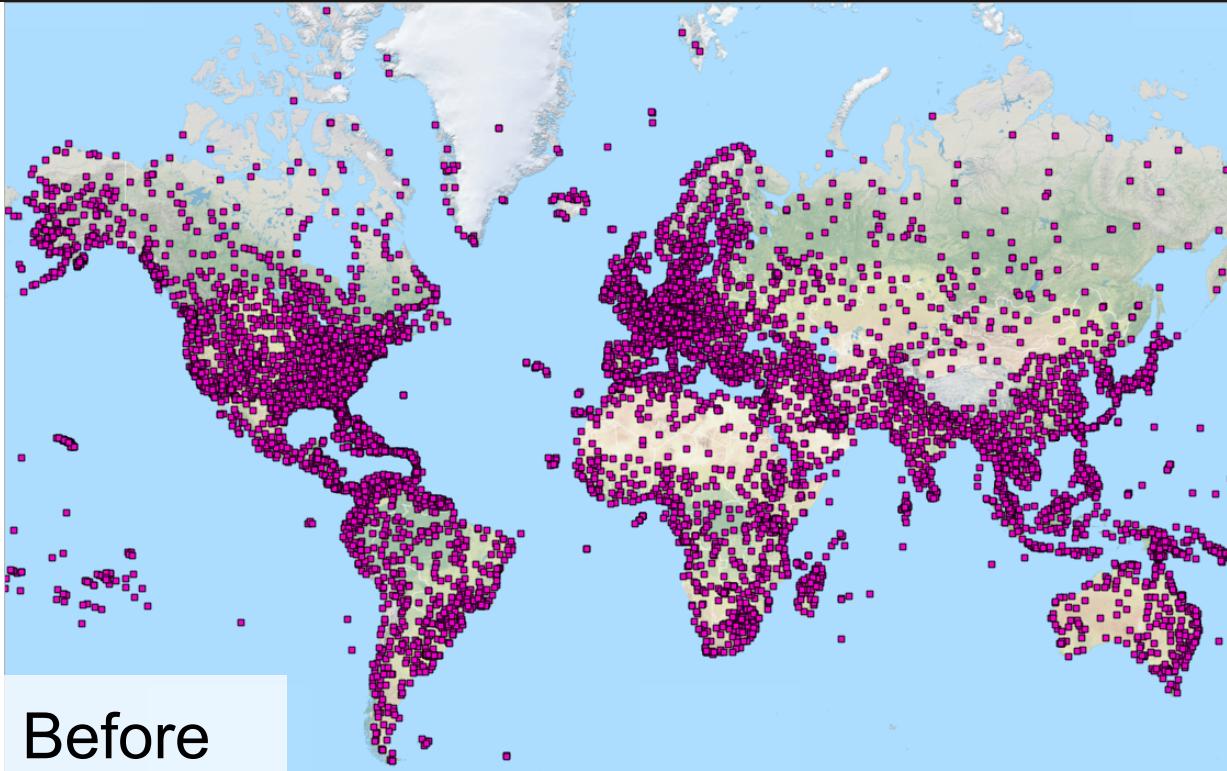
## Selection (our work)



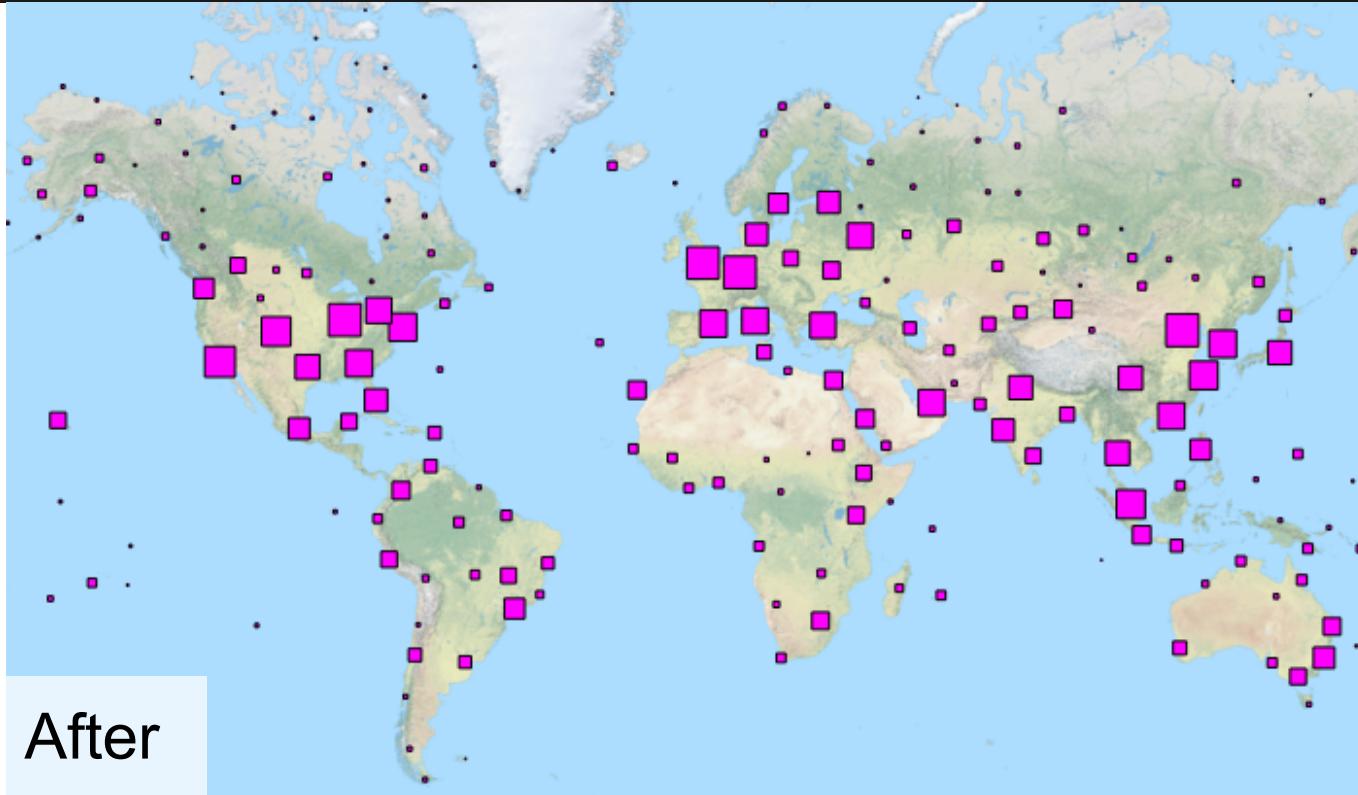
## Aggregation



# Selecting airports (example)



# Selecting airports (example)

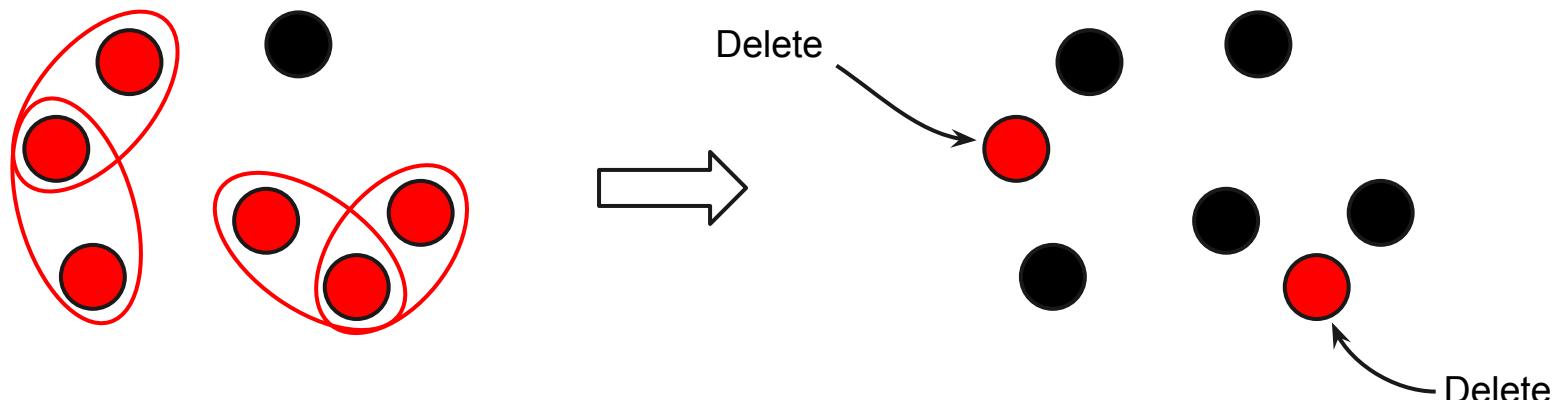


# Selecting airports (example)



# 1: Cartographic constraints

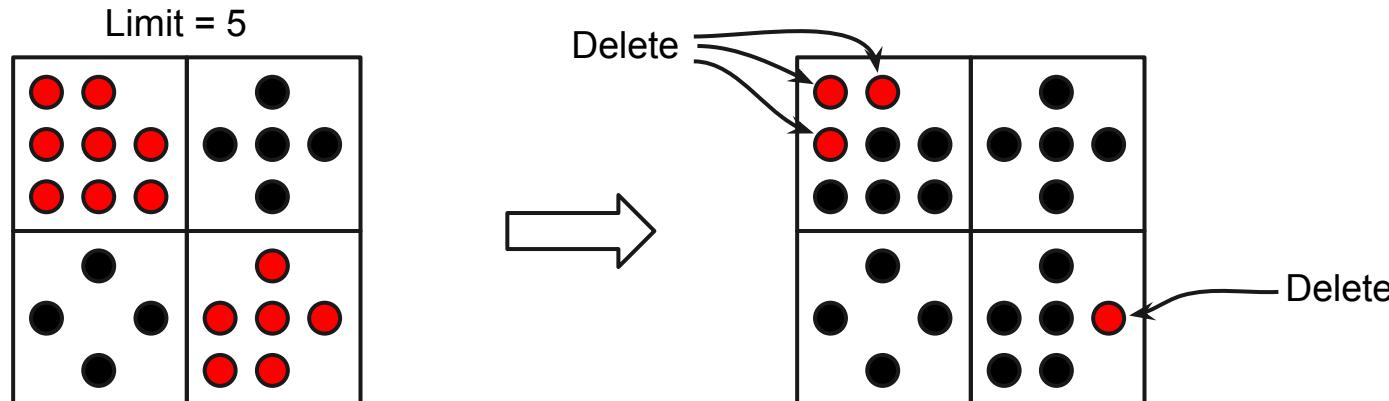
**Proximity constraint:** minimum distance between records  
(measured in pixels on screen)



Conflicts for proximity constraint

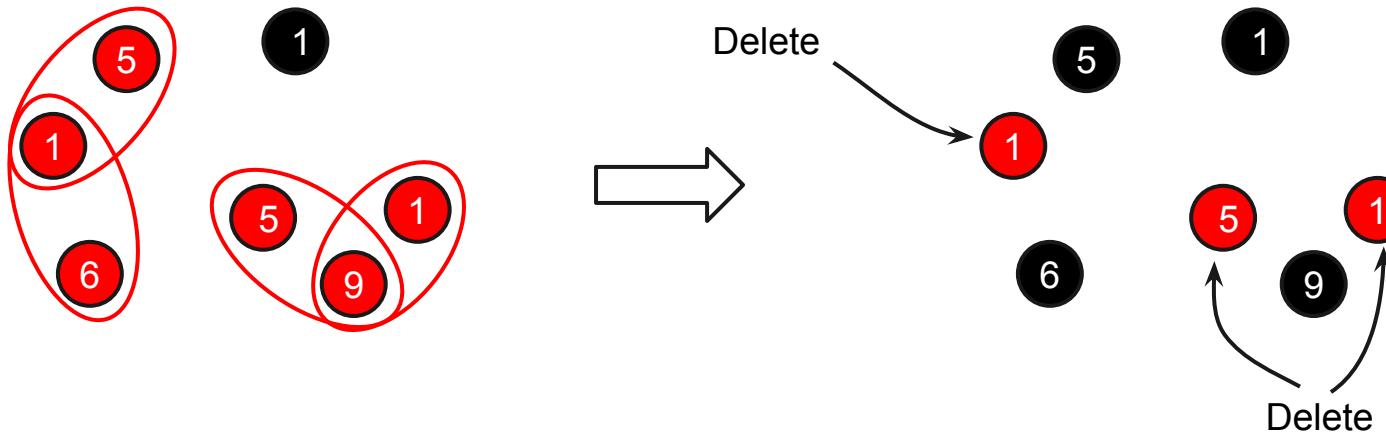
# 1: Cartographic constraints

**Visibility:** maximum records per unit area (within a map “tile”)



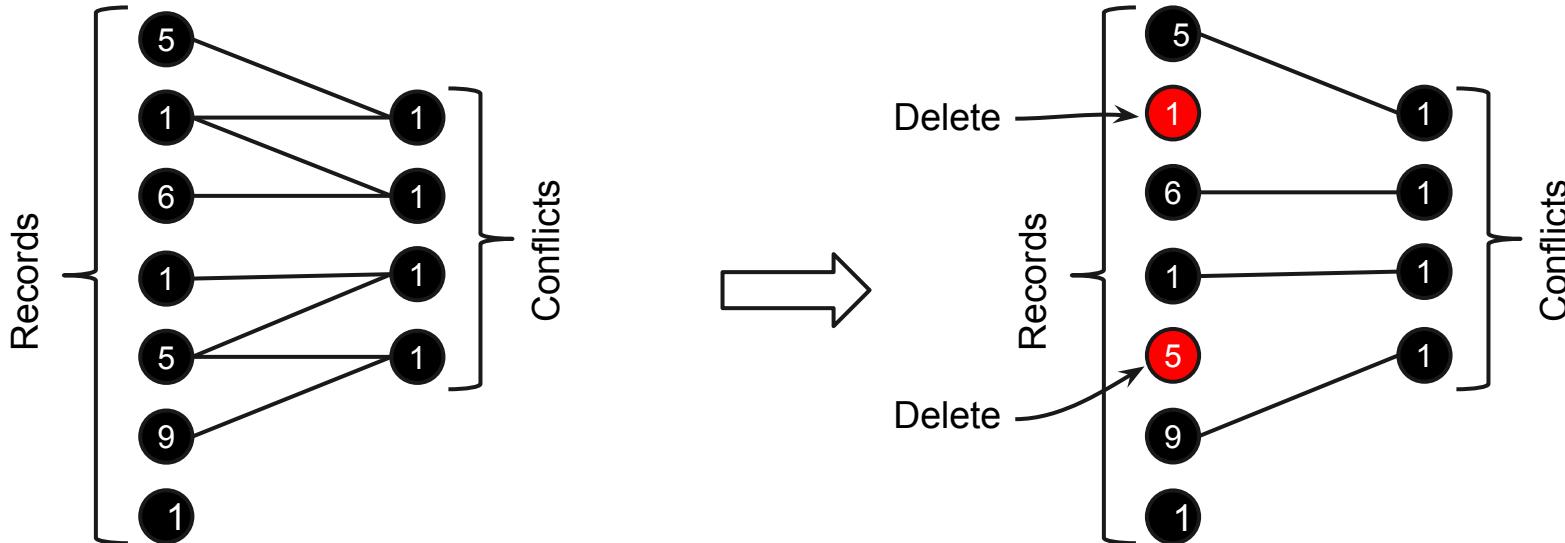
Conflicts for visibility constraint

# 2: Record weights



Adding weights may change selection

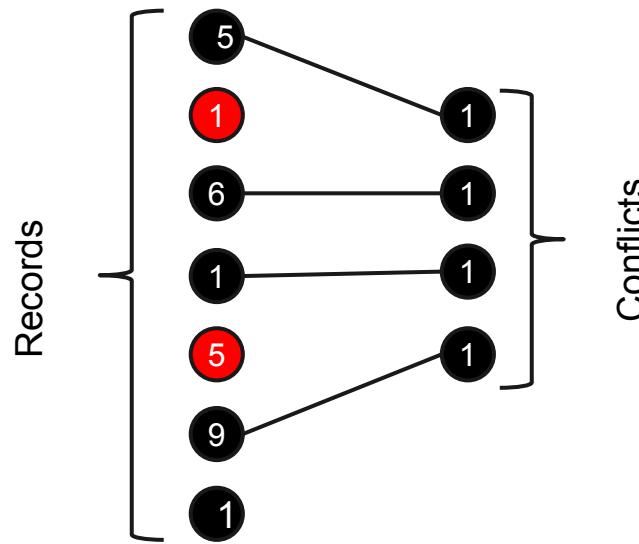
# Optimization problem



Problem: delete minimum weight cover

# Optimization problem

Set multicover problem  
(NP-hard)



# User control



Creating maps is the job of data-journalists, bloggers, high-level programmers...

... not mathematicians

# Declarative Cartography



**GENERALIZE**

airports **TO** airports2

**WITH ID** airport\_id

**WITH GEOMETRY** wkb\_geometry

**AT 18 ZOOM LEVELS**

**WEIGH BY**

num\_departures

**SUBJECT TO**

proximity 10

visibility 16

# Declarative Cartography

## Cartographic Visualization Language (CVL)

- Pronounced “Civil”
- State high-level goals:
  - Number of zoom-levels?  
**(AT 18 ZOOM LEVELS)**
  - Importance of records?  
**(WEIGH BY)**
  - Visual constraints?  
**(SUBJECT TO)**
- Compiles to SQL

**GENERALIZE**

airports **TO** airports2

**WITH ID** airport\_id

**WITH GEOMETRY** wkb\_geometry

**AT 18 ZOOM LEVELS**

**WEIGH BY**

num\_departures

**SUBJECT TO**

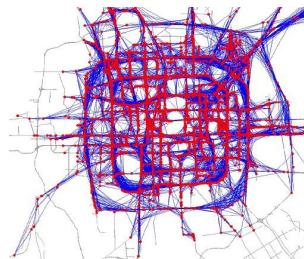
proximity 10

visibility 16

# Declarative Cartography

**Input dataset, e.g:**

- Points of interest (points)
- Running routes (lines)
- Animal territories (polygons)



**GENERALIZE**

**airports TO airports2**

**WITH ID airport\_id**

**WITH GEOMETRY wkb\_geometry**

**AT 18 ZOOM LEVELS**

**WEIGH BY**

**num\_departures**

**SUBJECT TO**

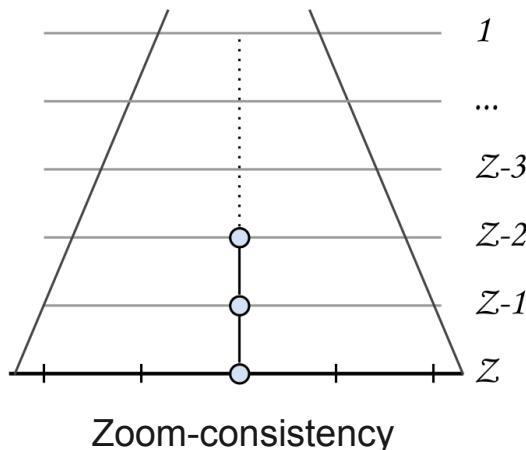
**proximity 10**

**visibility 16**

# Declarative Cartography

## Output dataset

- Output = input made “zoomable”



## GENERALIZE

```
airports TO airports2
```

```
WITH ID airport_id
```

```
WITH GEOMETRY wkb_geometry
```

```
AT 18 ZOOM LEVELS
```

## WEIGH BY

```
num_departures
```

## SUBJECT TO

```
proximity 10
```

```
visibility 16
```

# Declarative Cartography

## Weights

- Reuse SQL expressions as sub-language
- Evaluated for each row (once)

## Anything that is a number

- `random()`
- `x + y`
- `max (x, y)`

### GENERALIZE

```
airports TO airports2  
WITH ID airport_id  
WITH GEOMETRY wkb_geometry  
AT 18 ZOOM LEVELS
```

### WEIGH BY

```
num_departures  
SUBJECT TO  
proximity 10  
visibility 16
```

# Declarative Cartography

## Constraints

- Referenced by name
- Parameterized

### GENERALIZE

```
airports TO airports2
```

```
WITH ID airport_id
```

```
WITH GEOMETRY wkb_geometry
```

```
AT 18 ZOOM LEVELS
```

### WEIGH BY

```
num_departures
```

### SUBJECT TO

```
proximity 10
```

```
visibility 16
```

# Declarative Cartography

## Constraints

- Referenced by name
- Parameterized

## How are constraints defined?

- Let's look at proximity next

### GENERALIZE

```
airports TO airports2
```

```
WITH ID airport_id
```

```
WITH GEOMETRY wkb_geometry
```

```
AT 18 ZOOM LEVELS
```

### WEIGH BY

```
num_departures
```

### SUBJECT TO

```
proximity 10
```

```
visibility 16
```

# Declarative Cartography

## General syntax

- **CREATE CONSTRAINT**: give the constraint a name
- **NOT EXISTS**: define what a conflict is (select cid, rid)
- **RESOLVE IF DELETE**: define how to resolve a conflict (number of records to delete for conflict)

```
CREATE CONSTRAINT proximity AS
  NOT EXISTS (
    /* select cid, ric ... */
  )
  RESOLVE cid IF DELETE (
    /* scalar query or integer */
  )
```

# Declarative Cartography

## General syntax

- **CREATE CONSTRAINT**: give the constraint a name
- **NOT EXISTS**: define what a conflict is (`select cid, rid`)
- **RESOLVE IF DELETE**: define how to resolve a conflict (number of records to delete for conflict)

```
...
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
        ST_Distance(
            l.{geom}, r.{geom}) <
            CVL_Resolution({z}, 256) * {param_1}
)
...
...
```

# Declarative Cartography

## General syntax

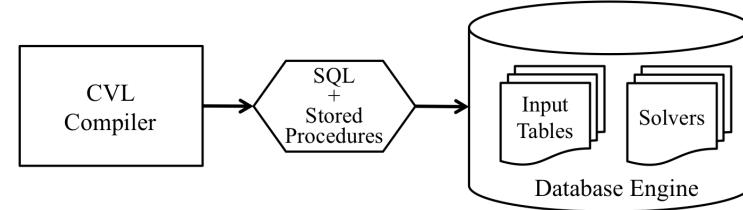
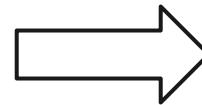
- **CREATE CONSTRAINT**: give the constraint a name
- **NOT EXISTS**: define what a conflict is (select cid, rid)
- **RESOLVE IF DELETE**: define how to resolve a conflict (number of records to delete for conflict)

...

```
RESOLVE cid IF DELETE (
    1
)
```

# Compiling CVL into SQL

```
CREATE CONSTRAINT proximity
AS
NOT
CREATE CONSTRAINT visibility
AS
)
NOT GENERALIZE
    airports TO airports2
    WITH ID airport_id
    WITH GEOMETRY wkb_geometry
    AT 18 ZOOM LEVELS
)
RESOURCES
)
RESOURCES
)
WEIGH BY
    num_departures
SUBJECT TO
    proximity 10
    visibility 16
```



# Using databases

- Spatial data typically managed in DBs
  - Option (a): Take data out of DB, process it, put it back
  - Option (b): Leverage database theory and technology to compute generalization *inside* DB (in-situ)
- We argue for option (b)
  - DBs are scalable and programmable
- However, DBs are not easy to program  
(unless you use CVL!)

# Results

Tested two algorithms

- Static Greedy Algorithm (SGA):
  - sorting based heuristic, implemented in SQL
- Linear Programming Greedy Algorithm (LPGA):
  - linear programming approximation algorithm,  
implemented in PLPython and external LP solver

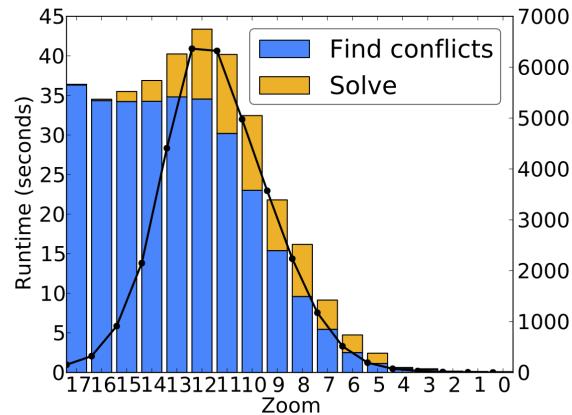
# Results

Tested with one RDBMS

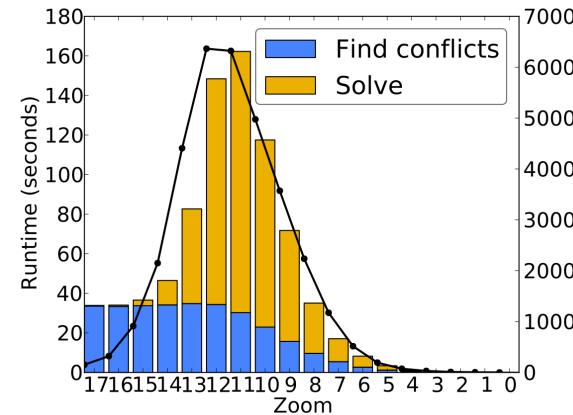
- PostgreSQL + PostGIS + Python + CVXOPT
  - Could be any database, e.g. a parallel one!

# Results for points

- 500K points, tourist POI from OpenStreetMap



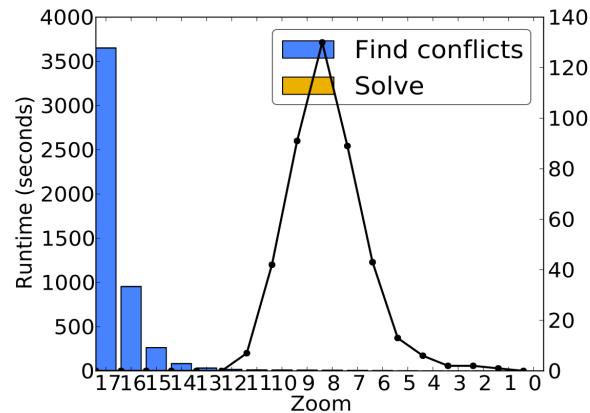
SGA + Visibility constraint



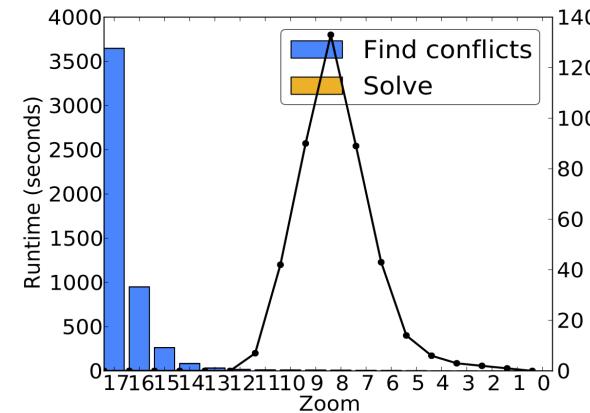
LPGA + Visibility constraint

# Results for complex shapes

- 4K lines/4M points, rivers from OpenStreetMap



SGA + Visibility constraint



LPGA + Visibility constraint

# Conclusion

- Core idea
  - Compute: *what* data to display on a zoomable map
  - Delay decision on *how* to display it
- Conclusion on results
  - Overall: Running time dominated by computing joins when using efficient heuristic for selection
  - Points: SQL-based algorithms perform best
  - Complex shapes: Joins for complex shapes run *much* slower than joins for points

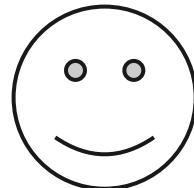
# Conclusion

- Pros:
  - Domain Specific Languages make databases easier to program
  - Database engines can compute a concrete selection that matches a high-level goal expressed in DSL
  - Databases can work on data in-situ
- Cons:
  - Running time must improve
  - Selection dependent on algorithm

# Conclusion

- Future work

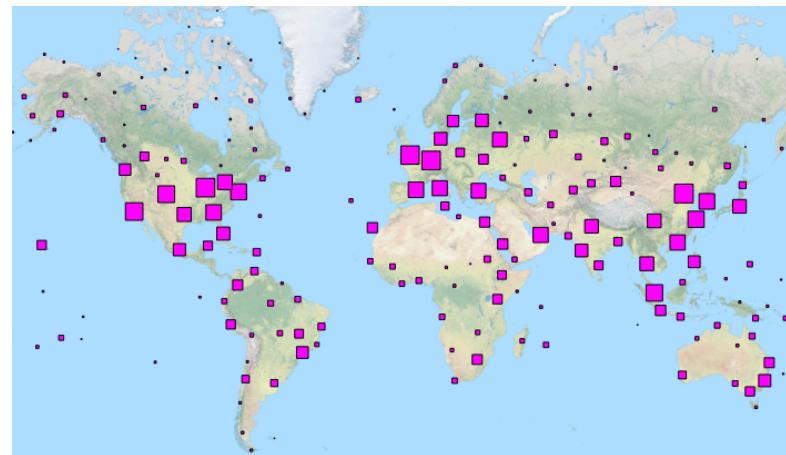
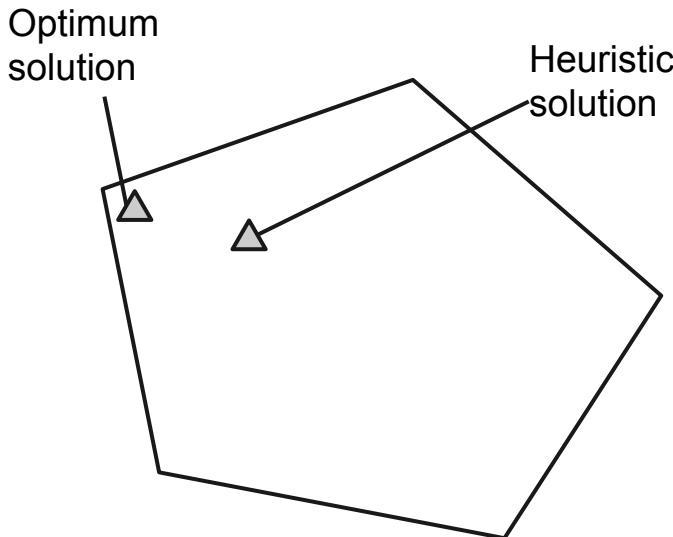
- Improving running time (in-memory, parallelism)
- Improve salience function (static weights too crude)
- Improve syntax (constraints are hard to define)
- Improve expressibility (e.g. relax zoom-consistency)



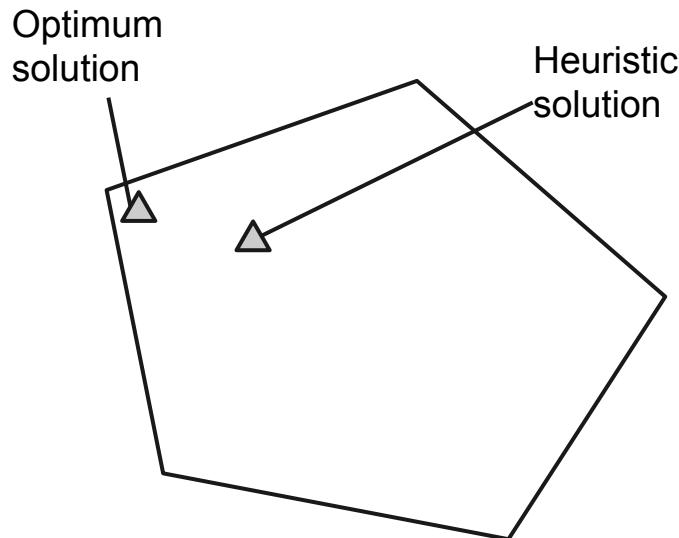
Thank you

# BACKUP SLIDES

# Algorithm matters!



# Algorithm matters!



Which airport to keep?

