A rule-based geospatial reasoning system for trip price calculations

Stefan Schenk

Supervisor: Willem Brouwer

Advisor: Mewis Koeman

Department of Software Engineering
Amsterdam University of Applied Sciences

This dissertation is submitted for the degree of
Bachelor Software Engineering

April 2018

Todo list

Table of contents

Chapter 2

Encoding Locations

2.1 Introduction

Encoding of locations has historically been of great importance, and is always being modernized. This chapter explains the general definition of locations, which types of locations are important for this project, how to make sure that the representation of these locations are universally interpretable, and how the encoded locations are to be stored in a database.

2.2 A Brief History Of Geographic Locations

A location is roughly described as a place or position. Throughout history, various navigational techniques and tools like the sextant, nautical chart and marinner's compass were used, measuring the altitude of the North Star to determine the latitude ϕ, in conjunction with a chronometer to determine the longitude λ of a location on the Earth's surface. The combination of coordinates is a distinct encoding of a location.

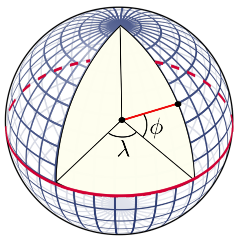

Fig. 2.1 A perspective view of the Earth showing how latitude and longitude are defined on a spherical model.

The history of this encoding goes way back to when it was first proposed in the 3rd century BC by Eratosthenes. He invented the discipline of geography, and was known for also being the first person to calculate the circumference of the Earth with remarkable accuracy. Today, navigation relies on sattelites that are capable of providing information to determine a location with an accuracy of 9 meters. The precision of hybrid methods using cell towers and Wi-Fi Location Services enables precise tracking of modern devices, still using the same encoding. Addresses are another representation of a location used in navigation. Addresses are easier to communicate than a pair of GPS coordinates, but can be ambiguous, imprecise, inconsistent in format. Addresses commonly make use of Postal Code systems, which have reliably been assigned to geographical areas with the purpose of sorting mail. Although even today, there are countries that do not have a Postal Code system. In contrast to the geographic coordinate system, postal codes describe streets and areas of varying shapes and sizes. A location being roughly described as a place or position, can be decomposed as an abstract term to describe physical or imaginary areas with varying radiusses and shapes. You could prepend 'the location of' to the following terms as an example: America, the birthplace of Sokrates, Wall Street, the center of the universe, the Laryngeal Nerve of the Giraffe, churches in the Netherlands. The final example presents the main challenge of this project, how to communicate a collection of areas of differing shapes and sizes that may overlap?

2.3 Requisites of Location Types

While setting up a backlog for a project, a shared knowledge about the terminology used in the issues must be achieved in order to collaborate effectively. Words or symbols do not have an absolute meaning, and ambiguity of abstract linguistic terms should be combatted by agreements. In Appendix A an agreement was made on the terms "area" and "point". A point is a unique place expessed as a distinct coordinate pair. An area is a set of many points that is tightly packed together, forming a polygon. The point and shape can be translated to the addresses and postal codes in the current taxiID pricing system respectively. This approach has some problems:

1. Addresses can be ambiguous.
2. Addresses and postal codes can be imprecise.
3. Postal code systems are not uniform.
4. Some countries don't have postal code systems.

In contrast, geospatial datatypes The Open Geospatial Consortium (OGC) Simple Feature Access ISO 19125-1 [2], would provide unique and precise location definitions that are uniform and universal.

The functional requirements state that, in the new system, users should be able to select predefined locations from external sources.

Excel sheets of address and postal code combinations are hard to maintain. Collections of addresses and postal codes are hard to visualize and communicate

much like a puddle is made up of many drops of water. What is needed for this project, is a method to determine whether a drop

$$p = (x,y)|xtussen - 90en90, ytussen - 180en180$$

P

$A = \mathscr{P}(P)$

Finally, collections of these possibilities are allowed to describe the problematic "all churches in the Netherlands" example:

$C_p \subseteq P$

and "all counties in which the majority voted Trump" example:

$C_a \subseteq A$.

This way a location could either be an area or a point, with which all possibilities are covered, except sets of these elements. As stated in Appendix A, the definition of an area is precise, unambiguous and easy to use in compare in computer programs. A single point may match another single point if it's the exact same point. A point may be sitting on top of a line or is contained within an area. The only other option is the negation of these statements. Because use cases for lines will be non-existent, points and areas are the proper candidates for spatial queries.

A taxi company director wants to be able to set price or define discounts from or to a certain location. They would like to define prices based only on departure locations, or only on destination locations, or both. For example: 'to Schiphol, a trip should cost €10,-', or 'from van der Valk hotels, a trip should cost €5,-', or 'from van der Valk hotels to Schiphol, the km price should be €0,60'. In the current implementation, a record would be stored containing departure location, destination location and price for every combination, where locations were defined as zip codes. Instead, it would make sense to be able to reuse locations after they have been defined once.

2.4 Literature Review

What 3 words, a multi-award winning global addressing system, bases 3m x 3m squares, covering the planet, on a combination of three words.

Geospatial

Postal code

kks32: See if other people solved locations

2.5 Database Prerequisites

The database must be capable of determining whether a virtual perimeter contains a set of coordinates, more specifically, it must adhere to The Open Geospatial Consortium (OGC) Simple Feature Access ISO 19125-1 [2] and ISO 19125-2 [3], including spatial data types, analysis functions, measurements and predicates for this requirement, or have some comparable implementation. The scenario presented in image 2.2 should be replicable.

kks32: Speak of database specific example requirements

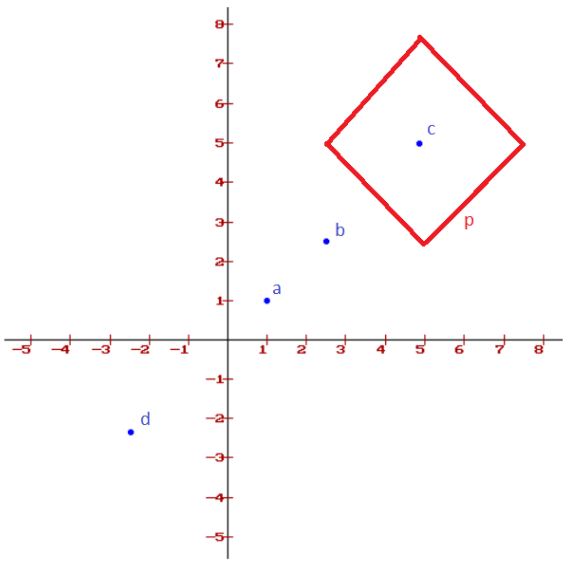

Fig. 2.2 Four Points, one Polygon p containing Point c.

2.5.1 OpenGIS Compatible databases

MYSQL's innate integrity is a good reason to opt for a full MYSQL database setup. MariaDB is a fork of MYSQL that performs better according to benchmarks, however they don't always translate to real life situations. It's easy to migrate from MYSQL to MariaDB, so choosing MYSQL at first could be preferable as an instance of MYSQL is already used at TaxiID. PostgreSQL offers a spatial database extender for that is OpenGIS compliant called PostGIS that adds support for geographic objects and location queries.

All spatial data types inherit properties such as type and spatial reference identifier (SRID). For rigorous documentation, both PostGIS documentation [4] and MYSQL documentation [5] could be consulted. When a generic geometry column, or point column is created, points can be inserted as shown in snippet 2.1 and 2.2

```
START TRANSACTION;
SET @a = ST_GeomFromText('POINT(1 1)');
INSERT INTO point (point) VALUES (@a);
SET @b = ST_GeomFromText('POINT(2.5 2.5)')
  ;
INSERT INTO point (point) VALUES (@b);
SET @c = ST_GeomFromText('POINT(5 5)');
INSERT INTO point (point) VALUES (@c);
SET @d = ST_GeomFromText('POINT(-2.5 -2.5)
  ');
INSERT INTO point (point) VALUES (@a);
# also insert @b, @c, and @d
COMMIT;
```

Listing 2.1 Insert four points

```
START TRANSACTION;
# First and last point must be the same
SET @a = PolygonFromText('POLYGON((2.5 5,5
   7.5,7.5 5,5 2.5,2.5 5))');
INSERT INTO polygon (polygon) VALUES (@a);
COMMIT;
```

Listing 2.2 Insert polygon

It is evident that c is contained in p. To determine which points are contained in p, the function as seen in Snippet

kks32: Add ref to snippet

can be used, which returns the point with coordinates $[5,5]$ as expected.

```
// All points contained in polygon
SELECT ST_ASTEXT(POINT)
FROM POINT
WHERE
ST_CONTAINS(
  (
    SELECT POLYGON
    FROM POLYGON
    WHERE id = 1
  ),
  POINT
)
```

Listing 2.3 Select points contained in polygon

```
// All polygons containing point
SELECT ST_ASTEXT(POLYGON)
FROM POLYGON, POINT
WHERE
  POINT.id = 3 AND ST_CONTAINS(
    POLYGON.polygon,
    POINT.point
  )
```

Listing 2.4 Select polygons containing point

2.5.2 OpenGIS Incompatible databases

MongoDB doesn't offer OpenGIS implementations but has geospatial query operators that may provide enough functionalities for current requirements

kks32: Add ref to Geospatial Query Operators — MongoDB Manual 3.6

. The argument for choosing one over the other depends on the vast differences between SQL and NoSQL, next to performance and extensiveness of geospatial features. The setup displayed in image

kks32: Add ref to image

is recreated in MongoDB using queries shown in snippet

kks32: Add ref to snippet

.

```
db.point.insertMany([
{ shape: { type: "Point", coordinates: [1,
   1] } },
{ shape: { type: "Point", coordinates:
  [2.5, 2.5] } },
{ shape: { type: "Point", coordinates: [5,
   5] } },
{ shape: { type: "Point", coordinates:
  [-2.5, -2.5] } },
])

db.polygon.insert({
shape: {
type: "Polygon",
coordinates: [ [ [2.5, 5], [5, 7.5], [7.5,
   5], [5, 2.5], [2.5, 5] ] ]
}
})

db.point.createIndex({ 'shape': '2dsphere'
   })
db.polygon.createIndex({ 'shape': '2
  dsphere' })
```

Listing 2.5 Select points contained in polygon

```
// All points contained in polygon
var p = db.polygon.find({})

db.point.find({
shape: {
$geoWithin: {
$polygon: [
[2.5, 5],
[5, 7.5],
[7.5, 5],
[5, 2.5],
[2.5, 5]
]
}
}
})

// All polygons containing point
var p = db.point.findOne({ coordinates:
   [5, 5] })

db.polygon.find({
shape: {
$geoIntersects: {
$geometry: {
type: "Point",
coordinates: [5, 5]
}
}
}
})
```

Listing 2.6 Select points contained in polygon

Next to database solutions for this requirement, services exist that are capable of geofencing. Although these services may not be free, and the added dependencies restrict extensibility.

2.6 Performance and Clustering Trade-offs

Agarwal and Rajan state that NoSQL take advantage of cheap memory and processing power, thereby handling the four V's of big data more effectively, but lack the robustness over SQL databases

kks32: Add reference to Agarwal and Rajan

. The report dives deeper into spatial queries and concludes that their tests suggest that MongoDB performs better by an average factor of 10, which increases exponentially as the data size increases, but lack many spatial functions that OpenGIS supports. Although improvements have been made

kks32: Add reference to Geospatial Performance Improvements in MongoDB 3.2," MongoDB

after the cited paper Schmid et al. 2015

kks32: Add ref to Stephan Schmid Eszter Galicz

was published. The team argues that clustering is much easier in MongoDB, which may be important in the future when the company grows. As the required functionalities exist in both SQL and NoSQL, it is beneficial to opt for MongoDB for its performance and alignment with the teams experience. Although if robustness is desired, or extra GIS functionalities required, SQL should be taken into consideration.

In what way can locations be represented to be universally interpretable?

1. Which types of locations should be distinguished?

2. What are the main differences between postal systems used around the globe?

3. Can postal codes be abstracted to geospatial data while retaining the same usefulness in the system?

4. How can different types of locations be effectively stored in a database?

References

[1] U. T. Inc. (2018) The uber story. https://www.uber.com/en-NL/our-story/. [Online]. Available: https://www.uber.com/en-NL/our-story/

[2] (2018) Simple feature access - part 1: Common architecture | ogc. http://www.opengeospatial.org/standards/sfa. [Online]. Available: http://www.opengeospatial.org/standards/sfa

[3] (2018) Simple feature access - part 2: Sql option | ogc. [Online]. Available: http://www.opengeospatial.org/standards/sfs

[4] (2018) Postgis 2.4.5dev manual. [Online]. Available: https://postgis.net/docs/manual-2.4/using_postgis_dbmanagement.html#PostGIS_GeographyVSGeometry

[5] (2018) Mysql 5.7 reference manual - geometry class. [Online]. Available: https://dev.mysql.com/doc/refman/5.7/en/gis-class-geometry.html

List of figures

List of tables