Bachelor thesis at the institute of computer science
of the Albert-Ludwigs-University of Freiburg

Context-Aware Search Spaces for Queries with Spatial Relations

---

presented by Rick Gelhausen
on the 27th of august 2015

Chair
Algorithmen und Datenstrukturen

Chairholder
Prof. Dr. Hannah Bast

Supervisor
Dr. Sabine Storandt

Processing Time
1.06.2015 - 1.09.2015

DECLARATION

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.
I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Ort, Datum	Unterschrift

Zusammenfassung

Online-Kartendienste, wie Google Maps, erlauben Anfragen in natürlicher Sprache, wie "Hotels Freiburg". Diese Anfragen werden dann als "Hotels in Freiburg" oder "Hotels in der Nähe von Freiburg" interpretiert. In dieser Arbeit stellen wir zwei neue räumliche Relationen vor, "zwischen" und die Himmelsrichtungen, welche Anfragen wie "Tankstellen zwischen Freiburg und Stuttgart" oder "Hotels nördlich von Freiburg" erlauben. Wir analysieren viele verschiedene Anfragen und entscheiden welche davon in einem gewissen Kontext Sinn ergeben. So macht es Sinn nach "Städten zwischen Städten" zu suchen, nicht aber nach "Städten zwischen Ländern". Dann erstellen wir eine Toolbox die für jede kontextbezogene Anfrage ein Suchpolygon zurückgibt. In einem letzten Schritt vergleichen wir die Resultate der Toolbox mit unseren Erwartungen. Wir zeigen dass in fast all unseren Anfragen sinnvolle Resultate herauskommen und geben abschließend einige Möglichkeiten die die Zukunft bietet.

Abstract

Web-mapping services, like Google Maps, allow natural language queries, like "Hotels Freiburg". These queries are interpreted as "Hotels in Freiburg" or "Hotels near Freiburg". In this thesis, we introduce the two new spatial relation "between" and the compass directions, allowing queries like "Gas stations between Freiburg and Stuttgart" or "Hotels north of Freiburg". To achieve this, we analyse many different queries and decide whether they make sense in a given context. It might make sense to search for "cities between cities", whereas it makes less sense to search for "cities between countries". Then, we create a toolbox that returns a search polygon for every context-aware query. In a last step, we compare the results from our toolbox with our expectations. We show that nearly all of our queries return useful results and conclude what possibilities are open for future work.

Contents

1 Introduction

1.1 Motivation

Nowadays web mapping services like Google Maps [5] and similar services from Microsoft and Apple have become an indispensable part of our daily lives. These services allow natural language queries like “Hotels Freiburg”. Users can search for hotels, supermarkets, train stations, restaurants and more. The spatial relations used are limited to “near” or “in”. “Hotels Freiburg” is interpreted as “Hotels in Freiburg” or “Hotels near Freiburg”. For Google Maps these spatial relations are implemented by setting the zoom to a certain level and showing all the hotels. Many spatial relations cannot be used in any of these services. For example “gas stations between Freiburg and Offenburg”, “hotels north of Stuttgart” or “playgrounds inside Central Park” are not viable queries in any mapping service.
In this thesis, we introduce two new spatial relations, “between” and the compass directions. To realize this, we use the open-source project Open Street Map (OSM) [3] which offers a vast amount of data that developers can use to create new map based content. Our goal is to use this data to create context-aware search-spaces represented by polygons. These search-spaces can be used to limit the results of a natural language query. The new spatial relations allow users to further customize their queries and thereby improve the user-experience.
In the following, we describe which queries on the new spatial relations are context-aware by analysing the most logical queries for each spatial relation. Then, we show how we implemented most of the queries presented in the first part. In a last step, we compare the results to the initial ideas and draw conclusions.

1.2 Related Work

In their paper “A splitting line model for directional relations”, Kevin Buchin et al. [2] introduce a method to describe directions using splitting lines. This method could be used to describe the compass directions in our work. One disadvantage of this method is the missing boundary condition regarding the distance. Queries, like ”countries west of Germany”, would return countries like Mexico. This is technically true, but as we will explain later-on this result would not be context-aware. Furthermore, this method goes into too much detail for many different shapes. We try to create a method that works well for most countries regardless of their shape.

2 New Spatial relations

In order to show possible new spatial relations that could find use in everyday applications, we chose the spatial relations "between" and "direction of". Both of these spatial relations will be analysed throughout this thesis and used to show what kind of queries will return useful context-aware search spaces and which queries need special cases or do not return any useful information at all. All human behaviour described throughout this thesis is based on assumptions from a survey of a small group of people. To further improve the queries, a large-scale survey could be conducted to understand how other people think about the queries presented in this thesis.

2.1 The "between" relation

The spatial relation "between" is the most interesting of both presented spatial relations regarding the diversity of the possible queries. In this context "between" describes a polygonal search field enclosed by certain points of interest. There are many useful objects that can be searched for between two points, polygons or poly-lines, for example countries, gas stations, hotels, supermarkets, bus stops, public restrooms or mailboxes. Despite the fact that all these queries are useful, most of them are only viable in a certain context. This means that looking for mailboxes between two streets inside a city might be a great idea, whereas looking for mailboxes between two countries is not. In this chapter we will give an overview over possible queries and we will analyse the context-awareness of these queries.

2.1.1 Sensible Search Queries between Countries

Finding queries that fit the requirements for context-awareness "between two countries" is no easy task. The only queries that come to mind would be other countries, bodies of water or cities.

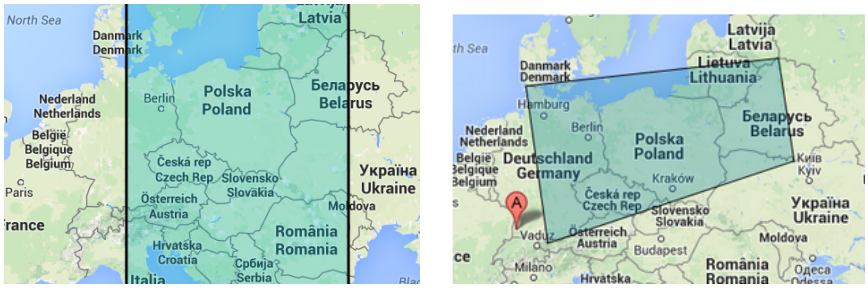
Bodies of water Even though checking for lakes or seas "between two countries" can be realized, there are several problems with this query. Many countries are not connected to seas or lakes and even if they are connected to seas, it does not imply that there is a sea between two of these countries. Germany and Italy are both connected to seas but there is only landmass between them.
Due to this query being a special case and the limited usefulness of the results, we do not consider this query to be viable.

Cities Cities, Hotels, Supermarkets and other similar objects are hard to place in a useful context regarding the query "between two countries". We could maybe interpret it as "on the border between two countries" for countries that share a common border. But most borders are very long and we cannot see any real usage for such a query, as we can see in *Figure 1*.

Figure 1: Hotels on the common border of Germany and Luxembourg.
This figure was taken from Google Maps and edited with Inkscape.

Countries Searching for countries "between countries" is the most obvious thing to do. There are several ways how to interpret "between" in the context of countries. It could mean that we draw horizontal or vertical lines and consider everything in between, like we did in *figure 2a*. This would mean that Egypt is between Germany and Ukraine. This is of course true but not in the context we would consider useful.
A better approach would be to consider the location of the countries to one-another and use certain points of both countries to create a polygon structure that represents the result set. For example Germany lies west of Belarus, combining the northernmost and the southernmost points of both countries forms a polygon containing Poland, Lithuania, the Czech republic, Slovenia, Austria and Denmark. Several of these countries are only partly contained in the polygon, as shown in *Figure 2b*. There are two ways to

refine the result set, calculating the centroid of each country and check if it is contained in the polygon or check if a country is inside the polygon by a certain percentage of area.



(a) (b)

Figure 2: (a) Trivial method using vertical lines. (b) Method using the orientation of countries

If one of the countries is giant in area and the other country small, for example Luxembourg and Russia, the result set becomes large but it still represents what we consider "between". *See Figure 3*

Figure 3: Countries between Luxembourg and Russia

There are several problems with this method. There are queries that do not make sense, for example "countries between USA and Vatican". This query is uninformative as both countries are far away from each-other, they are split by an ocean and their size difference is gigantic.

Another problem is that there are always two distances to consider because the earth is round. America and Russia are a lot closer to each-other when turning east from Russia, but due to the coordinate systems used in maps this would require a conversion of the coordinates. Fortunately, the line of coordinates where the negative and the positive coordinates meet is located in the ocean between Russia and America. The only country in the world that oversteps this coordinate line is Russia. As Russia is gigantic we can ignore the negative longitude coordinates for Russia without loss of precision. Due to this, there is no useful query that would require the conversion of the coordinates, as there is no country in between, *see Figure 4*.

Figure 4: Line where positive and negative longitude values meet.

2.1.2 Sensible Search Queries between Cities

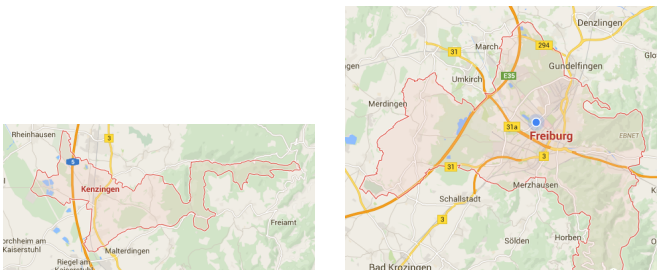
The polygon resulting from the search for something "between two cities" can vary depending on what we want to search for and on the distance between both cities.

Gas stations If somebody intends to look for gas stations "between two cities", it is likely that this person intends to drive from one city to another and wants to refuel on his way. Therefore it is best not to make a large area search. It is far more context-aware to calculate the shortest path between both cities and search around this path. To achieve this, it is possible to create a buffer around the points of the shortest path. This creates a tube

like polygon as represented in *Figure 5*.
If there are no gas stations in the result set, it is still possible to consider alternative routes to the other city.

Figure 5: Gas stations between Freiburg and Offenburg.
This figure was taken from Google Maps and edited with Inkscape.

Cities Searching for cities "between two cities" can be realized in two different ways. The first way would be to consider the city area. This method is similar to the method used to solve the query "countries between countries" and it would be the straightforward way to solve this query. Unlike country borders, city borders shapes are arbitrary, see *Figure 6*. For this reason it is probably better to have a method that works independent of the city borders.



(a) (b)

Figure 6: (a) City area of Kenzingen. (b) City area of Freiburg.
This pictures show how arbitrary the city borders can be and how they differ from the built-up area.

The second way is to ignore the city area and consider only one point of the built-up area of each city, preferably the city center. The city centres are then used to create an ellipse, where the city centres are the two points that are the furtherest away from the ellipse center, *see Figure 7*. Most people consider a city to be the built-up area and this ellipse approach describes this in a better way than the first approach.

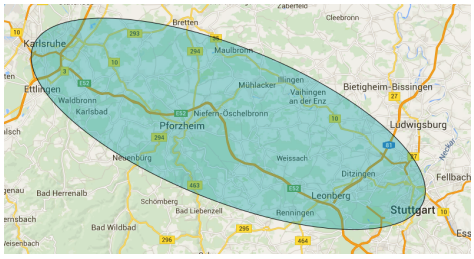


Figure 7: Cities between Stuttgart and Karlsruhe

Regardless of the method used, further restrictions are needed to ensure context-awareness. If the distance between two cities becomes too large, especially for cities that are not even on the same continent, this query becomes useless. If the distance between two cities is only 10 kilometres for example, we would include every city in the result set. The further the cities are away from one-another the larger the result set becomes. Depending on the situation, it might be a good idea to remove smaller cities or villages from the result set.

Hotels For hotels, there are two cases for searching "between cities" based on distance. For cities close to each-other, like Basel and Freiburg, it is more likely that you want to stay in the vicinity of those cities to stay in a hotel for a few days due to an event or just for sightseeing. This query is easily solved by using the ellipse method for cities, *see Figure 8*.

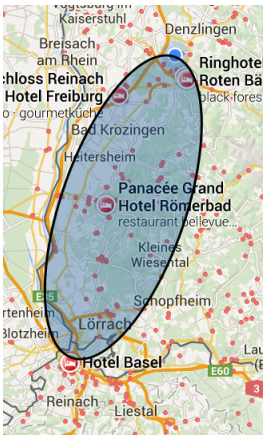
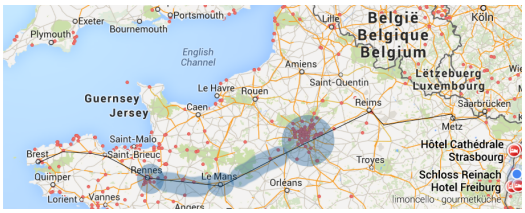


Figure 8: Hotels between Freiburg and Basel

This is not the case for cities that lie thousands of kilometres away from each-other, as shown in *Figure 9a*. The distance could easily be covered

in a one day trip. But it is advised to make an intermediate stop. If someone searches for hotels between two cities distant from each-other, they intend to drive there and have an intermediate stop somewhere along the road.
The ellipse method does not work here because the result set would contain tens of thousands hotels. A method to solve this would be to check for hotels along the street as we did for gas stations. To further improve this method we can check for cities along the path and include the hotels from those cities to the result set. We can describe this by using a set of circles. We can describe bigger cities with bigger circles to include all hotels. There is also a zone where search is irrelevant. Hotels that are too close to the starting city or the destination are both redundant as it is unlikely that people make stops early and nobody would stop driving when they are about to reach the destination, *see Figure 9b.*

(a)



(b)

Figure 9: (a) Shortest way between Brest and Saarbrucken. (b) Highlighted result set for this query.

Landmarks and historical buildings For historical buildings and landmarks it is best to use the ellipse method to cover a large area. Many interesting landmarks are often a bit of the streets and therefore it is better to search in a larger area. This query is only useful when searching between cities that are close to each-other. Figure 10 shows all historical buildings between London and Portsmouth. Both cities are at a perfect distance from one-another. The result-set is not too crowded and all the results are reachable when driving from London to Portsmouth. Some of the results might need a small detour, but this is acceptable as the point is to visit historical places in the vicinity.

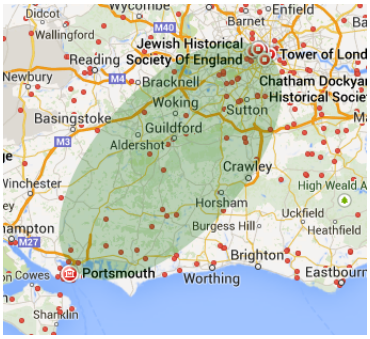


Figure 10: Historical buildings between Portsmouth and London

2.1.3 Sensible Search Queries between Streets

Streets are a complex dataset. There is an infinite amount of different shaped streets, some streets are long, some are short, some cross and others do not. There are distinctions between ways, highways, streets and many more.
The different types of streets are not really compatible. It makes no sense to look for something between a highway and a street. This is mainly due to the large amount of things that can be queried between two streets. You can look for buildings, like hotels or supermarkets, public transportation, mailboxes or public restrooms. Open Street Map [3] offers a large amount of different objects. It is best to search only between two streets of the same type. Streets of different types can have enormous differences in size, which leads to unreasonable result polygons. Ways are excluded for the most part as they are too small to lead to notable results.
There may be applications for search between highways for example “cities between highways”. For this work we focus mainly on streets inside a city because they allow the most interesting queries.
We can distinguish between two main types of roads. Roads that cross at

some point and roads that do not cross. For streets, the main focus is not to find different methods for different queries. It is about determining whether the streets cross or not and use different approaches for both cases.

Non-intersecting streets For non-intersecting streets the notion of "between" is easy to describe with a polygon. A simple method is to combine the endpoints of both streets. This forms a quadrilateral containing all the important objects, *see Figure 11.*

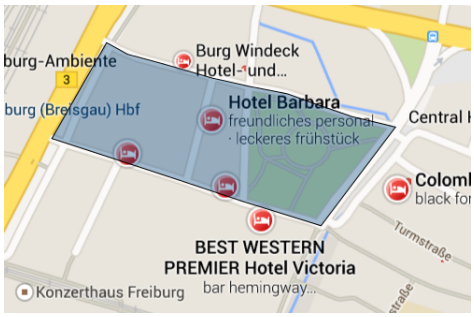
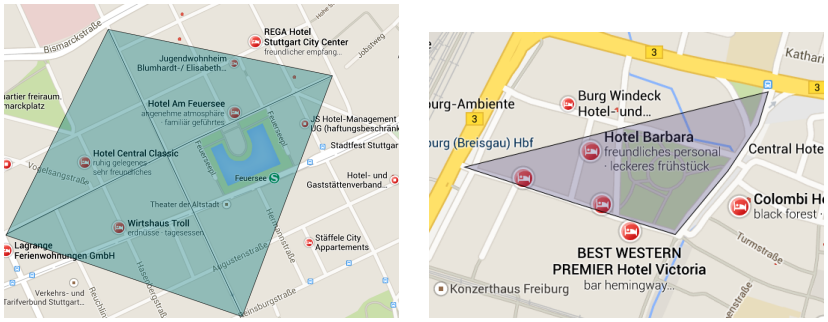


Figure 11: Hotels between two streets that do not intersect

The formed quadrilateral can take different shapes depending on the length and orientation of both streets.

Intersecting streets If two streets intersect, it is harder to define the notion of "between". Connecting the endpoints of the streets, without taking the other points into account, forms a quadrilateral that describes the notion best, *see Figure 12a*.
One special case are streets that intersect in one point. This special case can be solved by connecting the endpoints that are not part of the common corner. This forms a triangular shape representing the result set, *see Figure 12b*.



(a) (b)

Figure 12: (a) Hotels between two intersecting streets. (b) Hotels between two streets that have a common corner.

2.2 The Compass Directions

In this thesis, we will focus on "north of" as the main compass direction. All other compass directions, including North-west, South-west, North-east and South-east, can be created using rotation.
Compass directions need several restrictions. The query "hotels north of Freiburg" for example needs to be restricted to a particular region. Someone using this query does not want hotels from Norway in his result set, even though they are technically "north of Freiburg". This means that there has to be a rule for every query that determines exactly how far a search is context-aware.
For most people, "north" is not strictly restricted to 90 degrees from the equator. We situate "north" between 67.5 and 112.5 degrees.

2.2.1 Sensible Queries for Countries in a Compass Direction

For queries "north of countries" there is a variate of methods that can be used. The only useful queries in this particular case are countries. The first method is to draw a horizontal line through the northern-most point of a country considering everything above as "North".
The second method is to determine the western-most and eastern-most points of the country and find points at a certain distance to the north of these points. Then we rotate the new points around the initial points to form a trapeze structure, *see Figure 13*.

Figure 13: Countries north of Germany

The last method is a special method regarding the other methods used in this paper. We do not form a polygon but check for neighbouring countries using a dependency tree. This method has the advantage that we can decide by how many countries we move north.

2.2.2 Sensible Searches for Cities in a Compass Direction

There are again two ways to describe a city, either we take the built-up area or the real area of the city as our reference data. This might be dependant on the query but for the most part, people usually consider the built-up area as "the city".

Cities One way to determine "cities north of cities" is to use a similar method than the one used for countries.
The other method would be to create half of an ellipse and rotate it, *Figure 14* shows half of an ellipse that was rotated by 90 degrees.

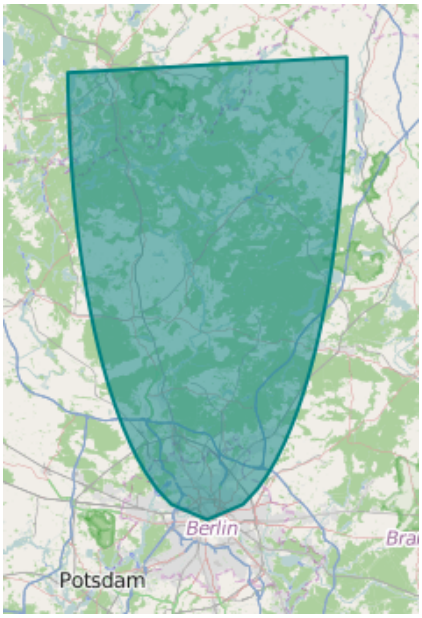


Figure 14: Hotels north of Berlin.

For both methods the search-radius should be determined by the area of the city. This is easy to understand when comparing Freiburg with New York City. Looking for cities north of New York City should yield a lot more results than for Freiburg.

Gas stations Looking for "gas stations north of a city" is a redundant query. If we want to drive somewhere to refuel, we also know our destination and we can check for "gas stations between two cities" instead. If we just

want to refuel inside a city, we can just use the existing spatial relation “near” used by many map services.

Landmarks, Hotels and Supermarkets For landmarks, Hotels and Supermarkets the idea is the same. We want to know if there is anything interesting in the north of this city. This leads to a large search area. These queries can be solved by applying the ellipse method introduced for the cities. For several cases especially for cities with huge built-up areas it might be better to use the city area.

2.2.3 Sensible Search Queries for Streets in a Compass Direction

For streets the orientation plays a role in eliminating useless queries. If a street is oriented from North to South the resulting polygon might not describe what is north of that street, *see Figure 15a.*
As for the spatial relation “between”, streets will be inside cities and allow for many different queries. There are no major differences between the queries regarding the resulting polygon. We have to determine a useful search area. For streets this can be made dependent on the street length.
A method to create the result polygon is to find a point in the respective direction for both endpoints of the street, combining these points will form a quadrilateral polygon, *see Figure 15b.*

(a) (b)

Figure 15: (a) Street that creates useless results for the query ”north of a street”. (b) Polygon describing everything ”north of a street”

2.3 Overview

In this section, we give an overview over the context-aware queries and by which polygon they will be represented. These are the results of what we analysed in the second chapter. The entries marked with an "x" are queries that are not considered context-aware. The first entry means that looking for a country between countries is context-aware and returns a quadrilateral.

"between"	country	city	street
country	quadrilateral	x	x
city	special case	ellipse	x
hotel	special case	ellipse or tube structure	polygon structure
supermarkets	x	ellipse	polygon structure
gas stations	x	tube structure	polygon structure
restrooms	x	x	polygon structure

compass directions	country	city	street
country	quadrilateral	x	x
city	x	semi-ellipse	x
hotel	x	semi-ellipse	polygon structure
supermarkets	x	x	polygon structure
gas stations	x	x	polygon structure
restrooms	x	x	polygon structure

3 Implementation

In this chapter, we will give a more detailed description of how we implemented most of the queries introduced in the first chapter. This chapter does not contain every query from the first chapter. It only contains the queries that were declared useful. Most of the data used in this chapter was extracted from Arcmap [1] using shape-files from Geofabrik [4]. Some data was taken directly from OSM [3].

3.1 Realizing Queries with "between" Relations

The spatial relation "between" offers clear boundary conditions which make the implementation easier than the compass direction queries. There are less variables like the search radii that have to be chosen manually or need a lot more information to determine.

3.1.1 A Toolbox for Search Queries between Countries

Country The first step is to determine the orientation of one country to the second country. To do this we calculate the centroid of both countries and determine their angle towards the latitude axis, as seen in *Figure 16*. Using the resulting angle, it is possible to distinguish eight cases, one for every compass direction. For each of the cases, different points of the given countries are needed. For example if one country is north-east of the other country, then we need the north-westernmost and the south-easternmost point of both countries to form the result polygon.

Figure 16

Once the orientation is clear, we determine the points we need. The southernmost point is the point with the lowest latitude, the northernmost point is

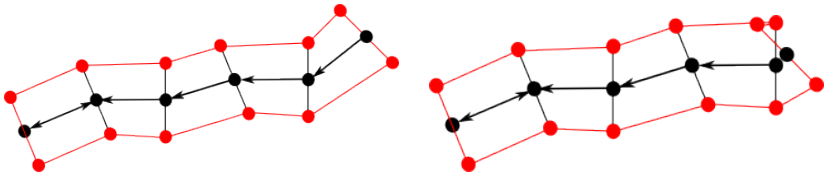
therefore the one with the highest latitude. The same works for the western and easternmost points using the longitude. The trickier part is to determine for example the north-westernmost point. To determine these special points we subtract or sum up latitude and longitude of every point. The point with the lowest difference in latitude and longitude is the north-westernmost point, the one with the highest difference is the south-easternmost point. The south-westernmost and north-easternmost points are determined by using the sum of longitude and latitude.
The last step consists of combining the points. Here we combine the two points with the same compass direction, then we combine the points from the country and repeat this one more time. This forms the quadrilateral result set, *see Figure 17.*

Figure 17

3.1.2 A Toolbox for Search Queries between Cities

Gas station This query requires the shortest path between two cities. We use a server running Open Street Route Machine(OSRM) [6] to achieve this. OSRM [6] provides a huge variety of functions. We mainly use the "viaroute" function, that returns the coordinates of the streets that form the shortest path between two given coordinates.
The goal is to create a tube structure. Our first approach was to always check the orientation between the next two points of the shortest path. Then we determine two points equidistant of the first point rotated towards the second point. We use the second point to determine the first point, *see Figure 18a.*

The downside of this method is that it can be hard to create a tube structure out of the result points. If the search-radius is bigger than the distance between two points the result points can overlap. Unfortunately, this happens often when working with streets, *see Figure 18b.*



(a) (b)

Figure 18: (a) Tube structure formed by combining points determined by the first method. (b) Errors created by the first method.

The method we apply now uses both a variation of the old method for distant points and circle intersections for points that are close to each-other. Instead of checking only the orientation towards the next point, we also take into account the orientation towards the last point and use the mean angle. Points that are too close to each-other are skipped as they do not change anything regarding the tube structure, *see Figure 19.*
This method also ignores the endpoints of the path. This does not matter as we are only interested in the gas stations between both cities and due to the large amount of points OSRM [6] delivers, the cities will still be represented in the result set.

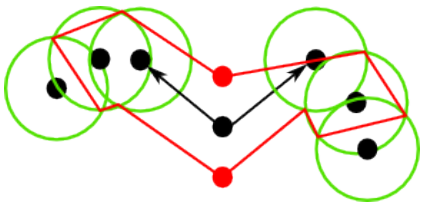


Figure 19

City For this query, we created an ellipse by using the distance between both city points. The distance is used as the major axis of the ellipse. We took half the distance as the minor axis. In a next step, we rotated the ellipse

using a rotation matrix.
This means that the city points will be the endpoints of the major axis, *see Figure 20.*

Figure 20: The structure of the rotated ellipse. Both cities are the endpoints of the major axis. d represents the distance between both cities. a is half of the distance and represents the length of the minor axis. b is half of a. Both variables a and b are then used to build the ellipse. This is how we imagined a nice ellipse, the variables can easily be changed to create different ellipse structures.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \tag{1}$$

Equation (1) represents the equation of an ellipse, where a is the length of the semi-major axis and b is the length of the semi-minor axis.

$$y = \sqrt{(1 - \frac{x^2}{a^2}) * b^2} \tag{2}$$

The ellipse equation can then be transformed to equation (2). This allows us to determine the y coordinates of the ellipse by using several x coordinates.

$$x' = x * cos(\theta) - y * sin(\theta) \tag{3}$$

$$y' = x * sin(\theta) + y * cos(\theta) \tag{4}$$

Using equation (3 & 4), every point is rotated by angle θ . θ represents the angle between both cities.

Hotels If the two cities are close to each-other, this query uses the ellipse function used for the cities. For cities further away from each-other, a method similar to the gas station method is used. It takes all cities on the shortest path between both cities into account. To accomplish this, we use a city grid. All cities are sorted into grid cells delimited by latitude and longitude values. As the street coordinates returned by OSRM [6] are very dense, we only use every tenth point to reduce the run-time and the amount of duplicates. Some

cities will be added more than once. These duplicates will be removed later on.
The city grid used should not contain small villages, they usually do not have hotels and are overloading the result-set. The result-set is represented by a set of circles, we could change their radius according to the city area.

3.1.3 A Toolbox for Search Queries between Streets

In chapter two we discussed the different types of streets. The first thing to do is to find out what type of streets are used. The two types were intersecting and non-intersecting streets. To determine which type of street is used, we implemented a function that determines if two lines intersect, inspired by a work from Martin Thoma [7].

Non-intersecting streets In the case of non-intersecting streets, we combine the endpoints of both streets to form a quadrilateral, including the points of both streets to maintain their shape. To ensure that the points are combined in the right order to form a quadrilateral, we would have to determine in which direction the street points of both streets are ordered.
A simpler approach is to combine the endpoints in both possible ways and check whether they would intersect. If they do not intersect they are combined correctly.

Intersecting streets If the streets intersect, we simply combine the streets endpoints by alternating between both streets. This forms a quadrilateral, ignoring the street points.

3.2 Realizing Queries with Compass directions

Queries about compass directions leave a lot of open variables, like the search-radius and the opening angle of the search polygon.

3.2.1 A Toolbox for Queries regarding Countries in a Compass Direction

The target for this query is a trapeze structure. For the example query “north of Germany”, we determine the eastern and westernmost points, as well as the northernmost point of Germany. We do this by using the methods used for the queries “between two countries”. Then we calculate the midpoint between the eastern and westernmost points and determine the distance from

this point to the northernmost point. *See Figure 21a.* This represents the length of the trapeze structure that will lie inside the country. We wanted to ensure that this distance does not influence the search-radius. The excess distance will be added to the search-radius.
The next step is the creation of the trapeze. It will be created by combining the eastern and westernmost points with two points in the north that will be calculated using the search-radius. In order to determine these points, we added the search-radius to the longitude of both points. *See Figure 21b.* Then these new points are rotated around the original points. The first point is rotated by 90 degrees plus an additional 15 degrees to create a typical trapeze shape. Analogous to the first point, the second point is rotated by 90 degrees minus 15 degrees, *see Figure 21c.*
Combining the points will form the result polygon, *see Figure 21d.*

3.2.2 A Toolbox for Queries regarding Cities in a Compass Direction

For this query, we chose to create half of an ellipse to describe the result polygon. We achieved this by using the ellipse function introduced for the queries "between cities". The search-radius could be set manually but there is a huge difference in size between many cities. The search-radius should therefore be determined using the city area.
We calculate the area of the cities using the city shapes extracted from OSM Boundaries [8]. The search area (radius) is then determined using the following formula.

$$radius = \frac{area * numberOfCities * \lambda}{\alpha} \tag{5}$$

The variable *area* represents the normalized area of the city we are interested in. In this case we normalized by dividing each area through the sum of every area. The variable *numberOfCities* represents the number of cities in our dataset. We apply this variable because the area decays fast with an increasing number of cities. This should help to slow the decay a bit to make the search for λ easier. λ is a constant. For our small test set, we used a λ of 400 to create acceptable results. Multiplying all these variables we get the search-radius in kilometres. To transform this to a distance on the map we divide it by α. In this case, α is 111.13, which means that 111.13 kilometres represent one unit on the map in terms of coordinates. The value used here is the average value for our planet.

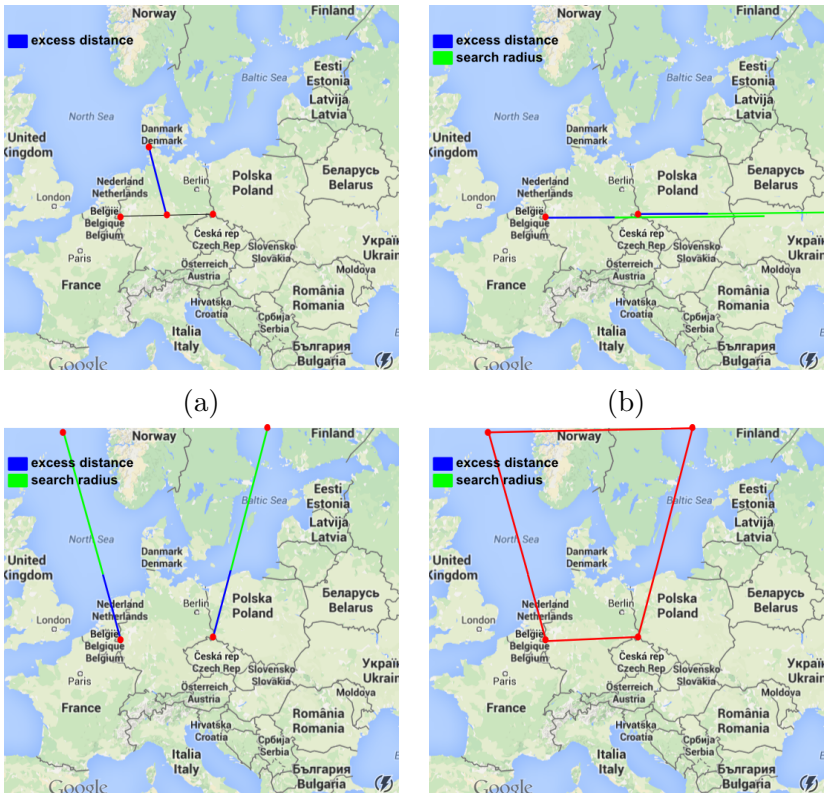


Figure 21: (a) Determine the excess distance using the extrema points of the country. (b) Determine two points that will be rotated to form a trapeze. (c) New points determined by rotating around the initial points. (d) The resulting trapeze structure.

3.2.3 A Toolbox for Queries regarding Streets in a Compass Direction

The method we use for the streets is similar to the one introduced for countries. The search-radius can be defined using the street length. We again find two points east of the two endpoints of the street and rotate them around the old points to create a quadrilateral. The orientation of every street makes several compass directions return useless results. We could add further restrictions, but due to the arbitrary shapes of streets this could exclude useful results. Our method includes all results.

4 Results

This chapter is a comparison between the last two chapters. We compare the ideas from the first chapter with the results from the implementation. As stated in the second chapter, our expectations are derived from a small survey. We asked the opinion of ten people, thus a large scale survey would be needed to verify our results. We tested each toolbox with about ten queries. We also tested special cases for each toolbox and show example results in this chapter.
In this work, our toolbox returns polygons as results. These polygons can then be used to create new applications. Google Maps [5], for example, seems to set a view-port and a zoom-level to describe their spatial relations, without creating a polygon.

4.1 The "between" Relation

The results for the "between" relation are generally good as it offers straightforward boundaries for most queries.

4.1.1 Results for Queries between Countries

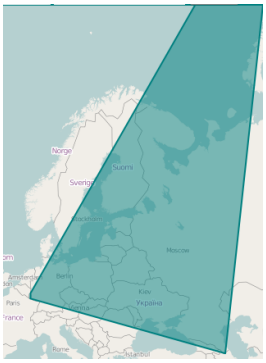
Country The results for countries are very satisfying. The intended result was a polygon that contains countries that are considered to be "between two countries".

(a) (b)

Figure 22: (a) How we imagined countries between Germany and Belarus. (b) The output of the implementation

As we can see in *Figure 22a*, the result we imagined contains the Czech Republic, Poland, Lithuania, Denmark, Slovakia, Austria, Ukraine and Russia. Only the Czech Republic, Poland and Lithuania would pass the area or centroid test. The rest of the countries do not have enough landmass contained

in the polygon.
The result from the implementation contains the same countries and looks exactly how we imagined it.
The method also works for countries that are separated by a body of water or queries between a small and a huge country. Even though the results might seem a bit strange for some queries, they are still valid for the most part and there is no reason to block such queries.



(a)

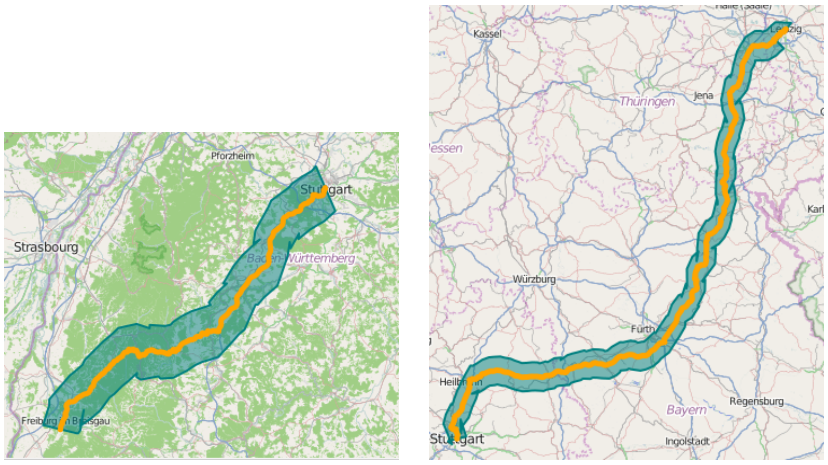
(b)

Figure 23: (a) Countries between Luxembourg and Russia. (b) Countries between Mexico and Algeria.

The result for "countries between Luxembourg and Russia" contains many countries and all of them can be considered "between" both countries, but there are some countries, like Romania, that may or may not be considered "between" Luxembourg and Russia, *see Figure 23a.*
The result for "countries between Mexico and Algeria" returns Cuba, Dominican Republic, Mauricio, Western Sahara and Morocco. These are all correct countries but it is questionable whether the query is useful with both countries being so distant from each other and split by the ocean, *see Figure 23b.*

4.1.2 Results for Queries between Cities

Gas stations The result for "Gas stations between cities" is a tube structure.

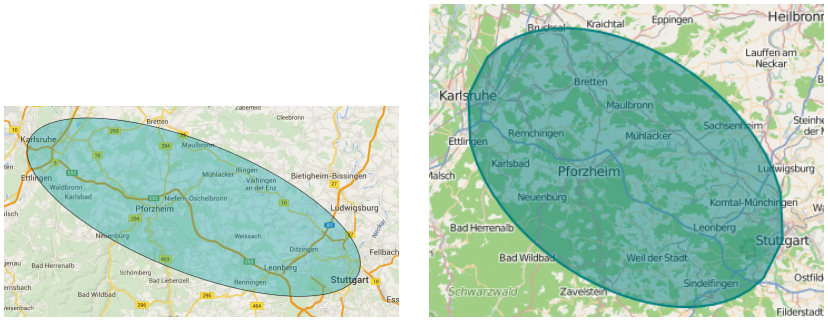


(a) (b)

Figure 24: (a) Gas stations between Freiburg and Stuttgart (b) Gas stations between Leipzig and Stuttgart

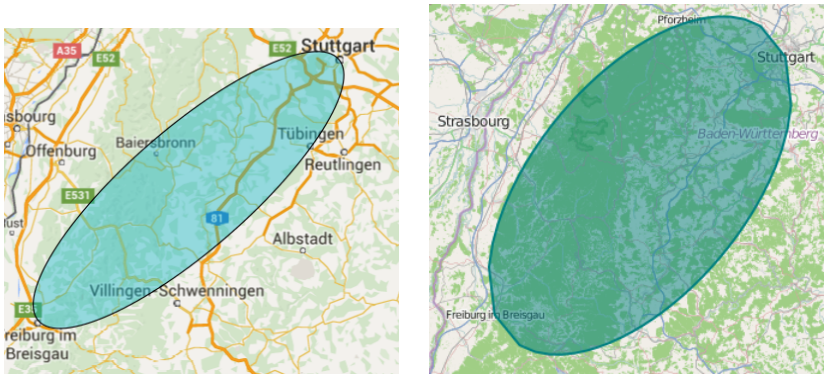
The goal was to create a tube-like structure around the shortest path between both cities, as most people who would ask this query want to go from the first to the second city. This could even be improved to allow for several paths using the "alternative path" feature of OSRM [6]. Like this the user could choose his favourite path and check for gas stations on that path. The results we got from this query are represented in *Figure 24*.

Cities The intended result for "cities between cities" is an ellipse. The results we received from our method are satisfying. The only flaw is a small inaccuracy created by rotating the coordinate points on the map. The ratio difference between latitude and longitude causes a small distortion, *see Figure 26.*



(a) (b)

Figure 25: (a) Cities between Karlsruhe and Stuttgart (Expectation)(b) Cities between Karlsruhe and Stuttgart (Implementation)



(a) (b)

Figure 26: (a) Cities between Freiburg and Stuttgart (Expectation) (b) Cities between Freiburg and Stuttgart (Implementation)

Open Street Map [3] offers many different types of cities like village, settlement, city or county. This allows us to further refine the result depending on the distance between both cities. The result set for "cities between Karlsruhe and Stuttgart" would not be too large if we except all kinds of cities, like

counties, villages and settlements, *see Figure 25b*.
Whereas the result set for "cities between Freiburg and Bremen" would be astronomical in size. Therefore we could only include major cities in the result set, *see Figure 27*.

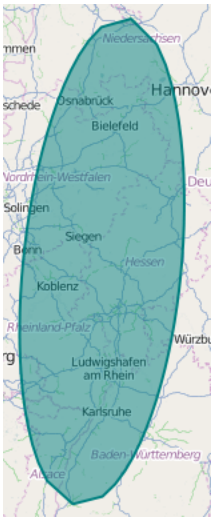


Figure 27: Cities between Freiburg and Bremen

Hotels The results for "hotels between two cities" are mixed. The results for cities close to each other are satisfying, apart from the small distortion, as we use the ellipse method, *see Figure 28*.

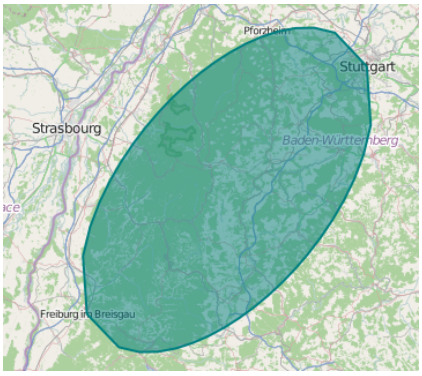
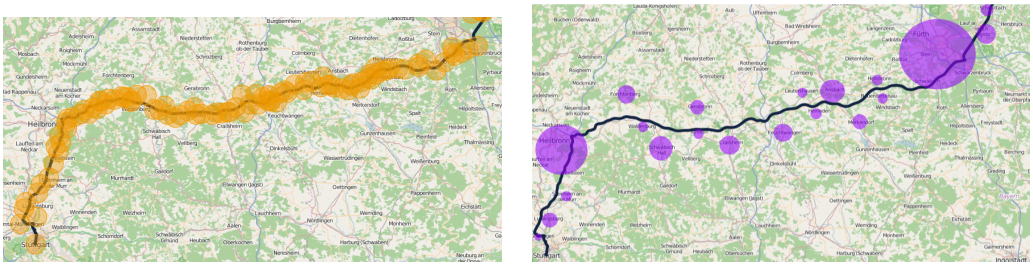


Figure 28

For cities that are distant from each-other, we applied a method to determine the cities along the shortest-path between two cities. In *Figure 29a*, we can see that our result looks a bit crowded. We used a data-set containing that contains all villages and cities of Germany. A data-set without small villages would be preferable.



(a) (b)

Figure 29: (a) Results we got for "hotels between Stuttgart and Berlin". (b) Results we could get for "hotels between Stuttgart and Berlin".

In *Figure 29b*, we can see how the result could look in the future. Not every village is taken into account and the search-areas are dependent on the size of each individual city.

4.1.3 Results for Queries between Streets

The results for "queries between streets" return the results we hoped for. The non-intersecting streets are taking the shapes of both streets into account. *See Figure 30.* Picture (a) and (b) show the results for non-intersecting streets. Picture (c) and (d) show the result for intersecting streets, where (c) is the special case with only three corner points. This method takes the areas between the street segments into account.
The "Berliner Allee" used in Figure 30b is already a very large street. But as it is still limited to the city, the results should still be useful for some people.

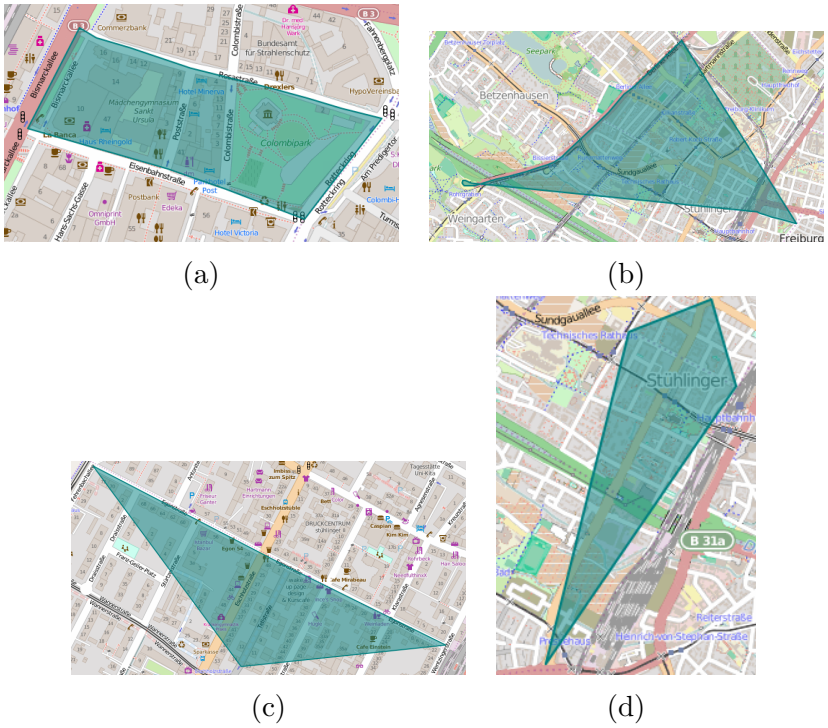


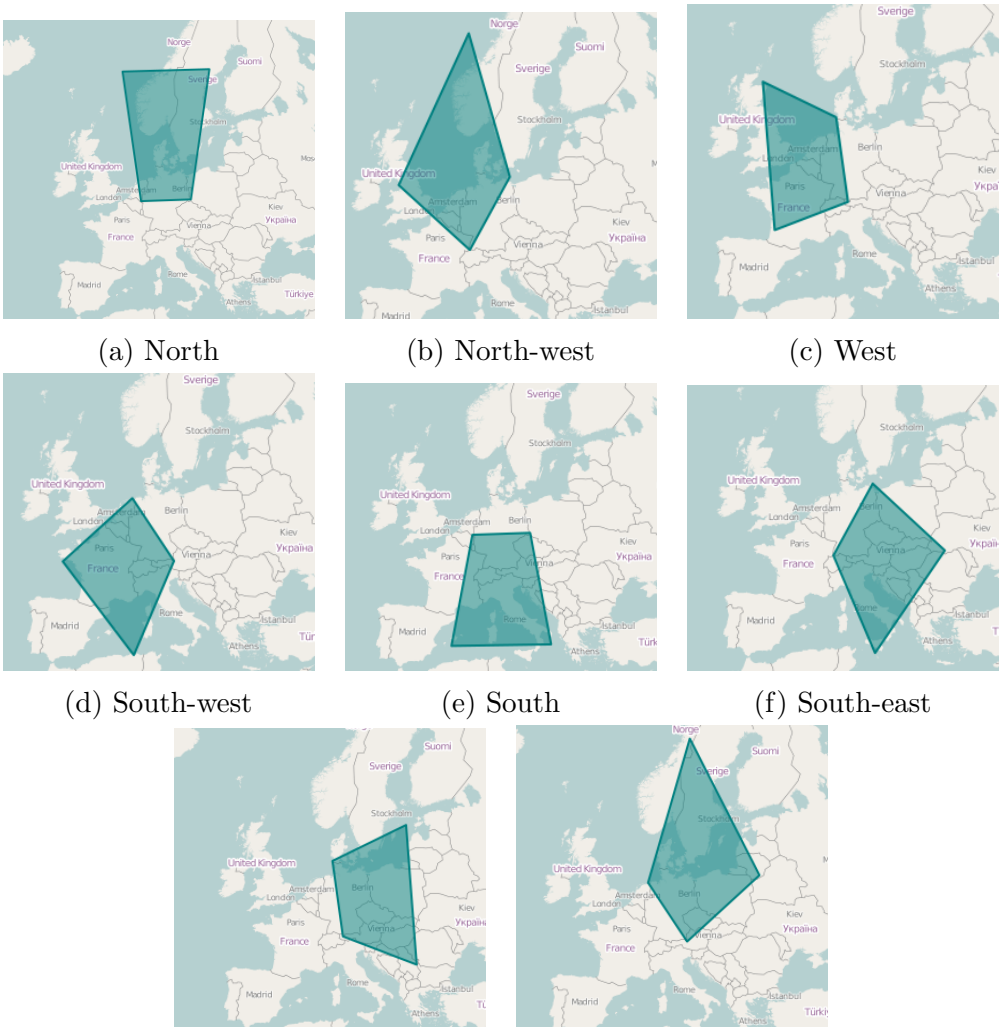
Figure 30: (a) Hotels between Eisenbahnstraße and Rosastraße. (b) Hotels between Berliner Allee and Eisenbahnstraße. (c) Hotels between Egonstraße and Tellstraße. (d) Hotels between Eschholzstraße and Egonstraße.

4.2 The Compass Directions

The results for compass directions have one common distortion due to the ratio difference between latitude and longitude. This leads to minor rotation errors and length differences. Those differences are less prominent for the cities and especially for streets. The smaller the result polygon, the smaller the mistake due to the distortion.

4.2.1 Results for Queries with Countries in a Compass Direction

The results for "countries north of countries" are not very satisfying when using the polygon method. For some countries and some direction it works fine, but due to the arbitrary shapes of the countries and the small distortion in the rotation it is hard to find relevant data in the average case.
See Figure 31. The main problem with this queries is that the results are arbitrary. For some directions it yields no results, for some directions there are many countries clumped together, and for some directions the search-space contains only a part of a country.
As mentioned in the chapters before, it would be better to use another approach for this queries. Using a neighbourhood-tree for the countries, it would be possible to have general results for every directions. We would not have to use distances using this method, we could use layers of countries instead.



(a) North (b) North-west (c) West

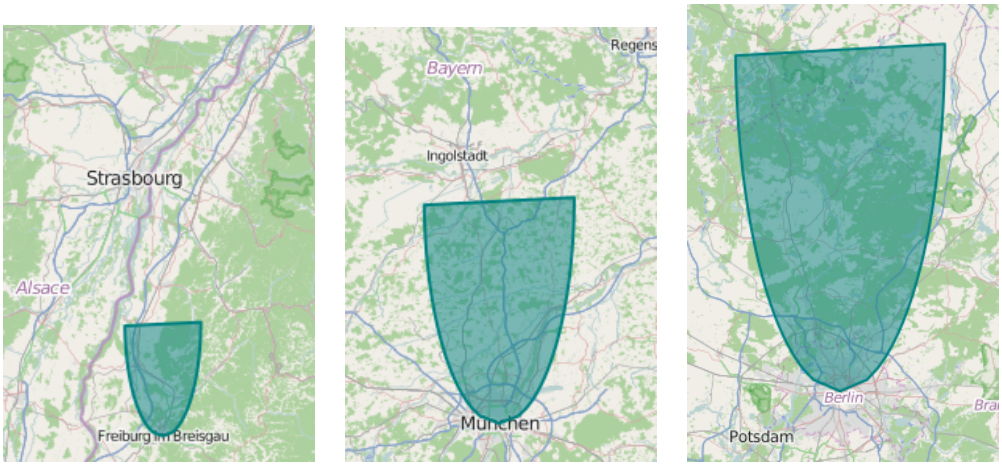
(d) South-west (e) South (f) South-east

(g) East (h) North-east

Figure 31: Compass Directions of Germany

4.2.2 Results for Queries with Cities in a Compass Direction

The results for queries in a direction of a city are satisfying. The size of the search-result increases with the size of the city, *See Figure 32*. The size of the result-set is only dependant on one constant λ. This allows us to customize it easily when we change the data set. One problem with this method is that we are using the administrative city borders. Some cities have a bigger area than others but the built-up area can still be bigger for the city with the smaller area. A better way to handle this would be to use the built-up area of the cities or to use a mix of our method and the number of inhabitants of a city. The built-up area has two downsides, it is hard to get data to calculate this area and unlike the administrative borders for cities the built-up area can change very quickly over the course of a year.



(a) (b) (c)

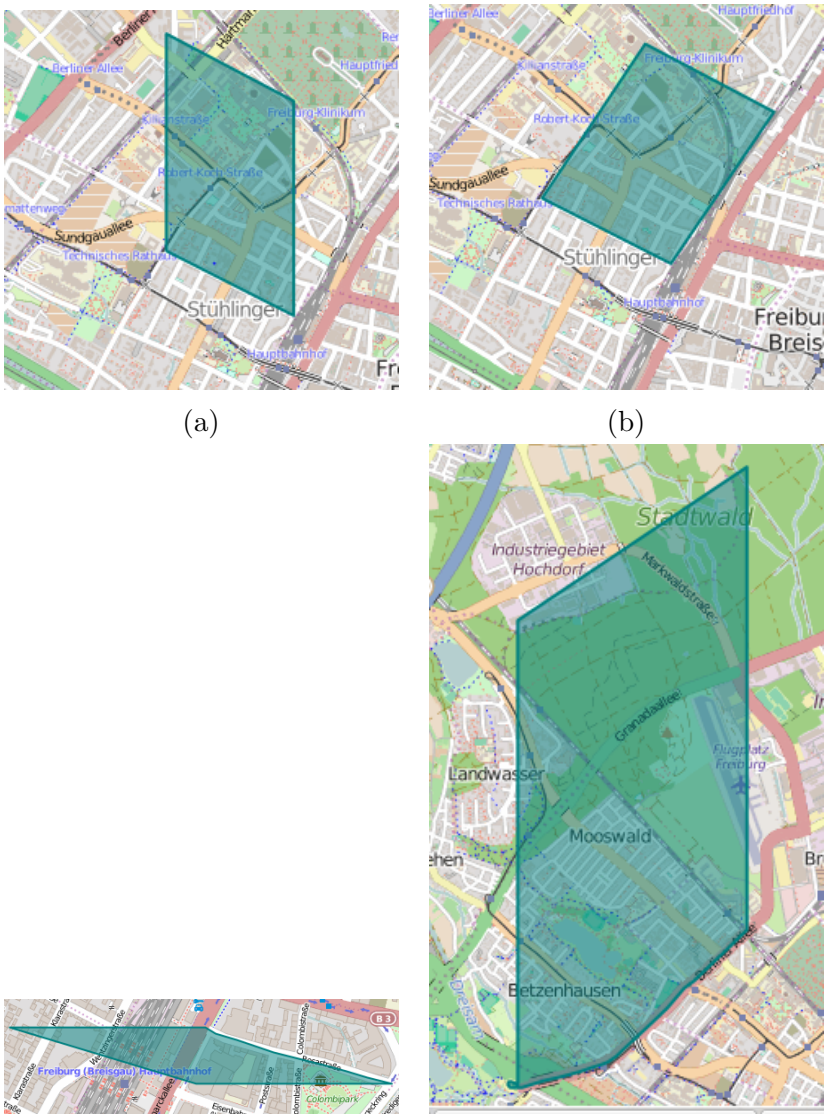
Figure 32: (a) North of Freiburg. (b) North of München. (c) North of Berlin. All the pictures were taken at the same zoom level to show the difference in size

The hardest part about these queries is to decide which λ gives us the most accurate representation of what we call "north of".

4.2.3 Results for Queries with Streets in a Compass Direction

The results for streets are how we imagined them. It is not hard to define a direction with streets. For each street there are several directions that do not make sense. We allow for these directions anyway because for some cases they return some useful results and it could be that the user is interested in these results, as he decided to ask this query.

Figure 33 shows us several possible queries. Figure 33a and 33b show two directions on a normal street. Figure 33c shows the result of a questionable query. The "Rosastraße" is directed in a way that looking west of the street does not make that much sense. But as we can see in the picture it, is still possible that there is a hotel or supermarket in the result-area and that the user searched for that particular thing. Figure 33d shows a query on a large street. It shows that the search-area changes with the street length. We used a ratio of 1.5 times the street-length to generate the figures. This ratio seemed the best to describe what we understood as "North of a street". If the ratio is too large, it makes no sense because no-one would walk several kilometres to get to a "supermarket north of a street", for example, and if the ratio becomes too small, the result-set may be empty.



(a) (b)

(c) (d)

Figure 33: (a) North of Egonstraße. (b) North-east of Egonstraße. (c) West of Rosastraße (d) North of Berliner Allee

5 Conclusion

Throughout this work, we described two new spatial relations, “between” and the compass directions. We first analysed the usefulness of several subqueries, then we implemented the most useful ones using OSM [3] and OSRM [6]. Finally, we described our results.
The spatial relation “between” yielded interesting results. The experiments with our toolbox showed that the search polygon returned by the toolbox matches our expectations in close to every case. Especially streets and countries offer many useful results that could be used to improve web-mapping services. Some improvements could still be done for queries that involve the calculation of shortest paths. We could, for example, include alternative paths to offer more possible results.
The compass directions proved a harder challenge than expected. There are many boundary conditions that have to be “guessed”, mainly the search-radius. Nevertheless, we have useful results for streets and cities. As stated in this work, the results for countries are usable but not great. It would be better to use a non-polygon method to solve these queries.
An overall improvement would be to remove the small distortion due to the earth shape, described in the results. As our work is a proof of concept, the performance can be further improved using e.g. databases. Furthermore, new spatial relations like “inside”, “along” or “around” can be created to further improve the possibilities a user has when using a web-mapping service.
As there is close to no research going into this field, regarding spatial relations for web-mapping services, we find it important to create new spatial relations. This is a way to create a new user-experience for web-mapping services.

References

[1] Esri ArcGIS. Arcmap, 2002. URL http://www.esri.com/software/arcgis.

[2] Kevin Buchin, Vincent Kusters, Bettina Speckmann, Frank Staals, and Bogdan Vasilescu. A splitting line model for directional relations. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 142–151. ACM, 2011.

[3] Open Street Map Foundation. Openstreetmap, 2004. URL http://www.openstreetmap.org/.

[4] Geofabrik. Openstreetmap data extracts, 2011. URL http://download.geofabrik.de/.

[5] Google. Google maps, 2004. URL https://www.google.com/maps/.

[6] Karlsruhe Institute of Technology. Open street route machine, 2011. URL http://project-osrm.org/.

[7] Martin Thoma. How to check if two line segments intersect, 2013. URL http://martin-thoma.com/how-to-check-if-two-line-segments-intersect/.

[8] TODO. Osm boundaries 3.2, 2015. URL https://osm.wno-edv-service.de/boundaries/.