Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

Thesis:

Interactive Computation of Isochrones

Bachelor in Computer Science and Engineering

Student: Benjamin Gruber
Student No. : 9139
Thesis Supervisor: Prof. Johann Gamper

Table of Content

Abstract

An isochrone is according to the Babylon online dictionary “a line drawn on a map connecting points at which something occurs at the same time”. The Free University of Bozen-Bolzano hosts a project about isochrones, called ISOGA (**Iso**chrones for **G**eo-Spatial **A**nalysis), to calculate the reachability of points on a map within a specific time frame. However, in its current version ISOGA lacks advanced interactivity. If the user of the program just wants to compute one explicit isochrone, he or she can simply input the required parameters and press “Compute”. But as soon as a more advanced interaction with the user is needed, it gets arduous: If somebody needs, for example, to compute many similar isochrones at once, there is no other way than to insert the variables for the first computation, press compute, insert the variables for the second computation, press compute and so on.

The present thesis aims to square up this problem by making isochrone queries more flexible and interactive. More specifically, this means that by inserting a slider mechanism the user will be allowed to alter values of the isochrone computation in an easier and more intuitive way: instead of reinserting or retyping numbers, it will now be possible to alter the size of an isochrone and its arrival respectively starting time by simply moving a slider. The new slider will be integrated into the existing program so that the user can use its preferred way of interaction, either the new or the old method. Furthermore, the problem of the single computation of an isochrone is also tried to be handled by inserting a mechanism, which allows the computation of multiple isochrones at once. It will be feasible to compute many isochrones at the beginning and refer to the received results, thus avoiding further calculations in some cases. The new method will use the already existing algorithms in an iterative way, which allows including the added functions with a minimal interference with the existing code.

1. Introduction

At first someone may ask what exactly an isochrone is, and as there are smarter people than me I will take their definition. An isochrone or, more precisely, an isochrone map is

"A line drawn on a map connecting points at which something occurs at the same time"
~Babylon online dictionary

We basically have one point in a field and can conclude through the use of isochrones which other points on the same field are exactly reachable in a fixed amount of time. Initially this sounds not very useful, since we usually want to know all the places where we can go in a fixed time and not only the places that we can reach in exactly the given time. But since we can conclude through the isochrone map which places are reachable precisely in the given interval and below, isochrones are – although highly CPU-intensive - extremely useful in the field of reachability analysis and route planning (in a simple way reachable points by one kind of transportation like feet, bike or car with a fixed amount of speed, or in a more complex way by using more transportation modes). Besides that, they are also used, for example, in the field of hydrology to measure the time water takes to reach a specific target like a lake or a reservoir when constant rain falls.

If we would have a constant speed to reach every point on an imaginary map, we would not need any computer scientists' help, since we could simply draw a circle around the point of origin and we would have exactly drawn all points that are reachable in a fixed amount of time. In a real world application like, for example, public conveyance, the speed could change heavily between more points if a point is reachable by foot, by bus or by train and some points may not even be reachable at all, such as the middle of a sea, so that the computation of isochrones becomes more and more complicated.

With all this given different information it is the algorithms' work to compute the fastest way by including all means of transportation. For example, the algorithm has to know how fast some points can be reached with all other possible means of transportation, if it could be reached by different means faster, how this would change if we increase the time, etc.

If we have only small maps with only few different options, the used algorithm seems not as important as it is, since a simple brute force technique (trying out all different paths from a given starting point) can give back the result in a short amount of time through the enormous power of today's computers, and even more their server counter parts. If we instead consider huge maps like, for example, the public transport network of a metropolitan area and increase the time, an efficient algorithm is critical for a fast and successful execution of this task. One algorithm to compute isochrones is currently[1] running at the Free University of Bozen-Bolzano, which has been developed in the ISOGA (**Iso**chrones for **G**eo-Spatial **A**nalysis) project **[1]**. The ISOGA project is able to compute the reachable points in a fixed amount of time by using bus or feet as transportation ways on a map of Bolzano, South Tyrol, Italy and San Francisco. This can be done by using three different algorithms and entering multiple parameters.

Of course this is not the only paper which handled isochrones or more specifically the ISOGA project. Gamper et al. present an efficient and scalable isochrone-computing algorithm,

[1] May/June 2014

called MINEX **[2]**, which keeps the memory requirement low by loading only relevant portions of the network into the RAM. Gamper et al. also formally define isochrones for multimodal networks with different transportation modes **[3]**. For their computations, which were also studied by Bauer et al. **[4]**, they used the same MINEX algorithm which was described before. Marciuska and Gamper elaborate how it is possible to determine the place of objects inside an isochrone **[5]**. Furthermore Innerebner et al. deal with how to design a web extension of a spatio-temporal data base management system like in the project which is mentioned in this paper **[6]**.

As an alternative to the ISOGA-isochrone-project, SimpleFleet[2] can be shortly mentioned, which focuses on spatio-temporal tracking. Efentakis et al. describe its isochrone computation algorithm with useful data to present its impact on live-traffic assessment **[7]**.

1.1 Aim

The overall aim of this thesis is to make the computation of isochrones more interactive and user-friendly. This will be done by showing how the university's isochrone project is designed and how we extended it with a slider mechanism to allow a user to compute a range of isochrones without specifying the input parameters for each individual isochrones. We analyse and discuss also how the computation of isochrones can be improved in the future by adding new elements or "*remaking*" the project as a lighter version with a minimal set of absolutely needed function, and possibly using new programming techniques that were not available when the project originally started. Generally the main focus will be on my own expansions of the isochrone project and how they were realized.

1.2 Structure of the Thesis

The thesis is structured into five chapters as follows:

- *Chapter 1* introduces the reader to the concept of isochrones and gives a projection of what the thesis will consist of and what the thesis aims for.
- *Chapter 2* describes the existing ISOGA isochrone project, which forms the basis for the described expansions of this paper and also its technical background.
- *Chapter 3* – the main part of this work – describes the newly made enhancements. It will consist of a description of the new additions (adding a slider mechanism, computing multiple isochrones at once), how this was done from a technical point of view and a running example.
- *Chapter 4* gives an overview about how the isochrone project can be further expanded and improved.
- *Chapter 5* gives final conclusions about what was achieved in this thesis.
- The appendix shows a short listing of which programs were used during this work and what the figures' sources are.

[2] **SimpleFleet** project web page: http://www.simplefleet.eu/

2. The ISOGA Project

The existing project that I extended during the work of my thesis was ISOGA, a system for geospatial analysis using isochrones.

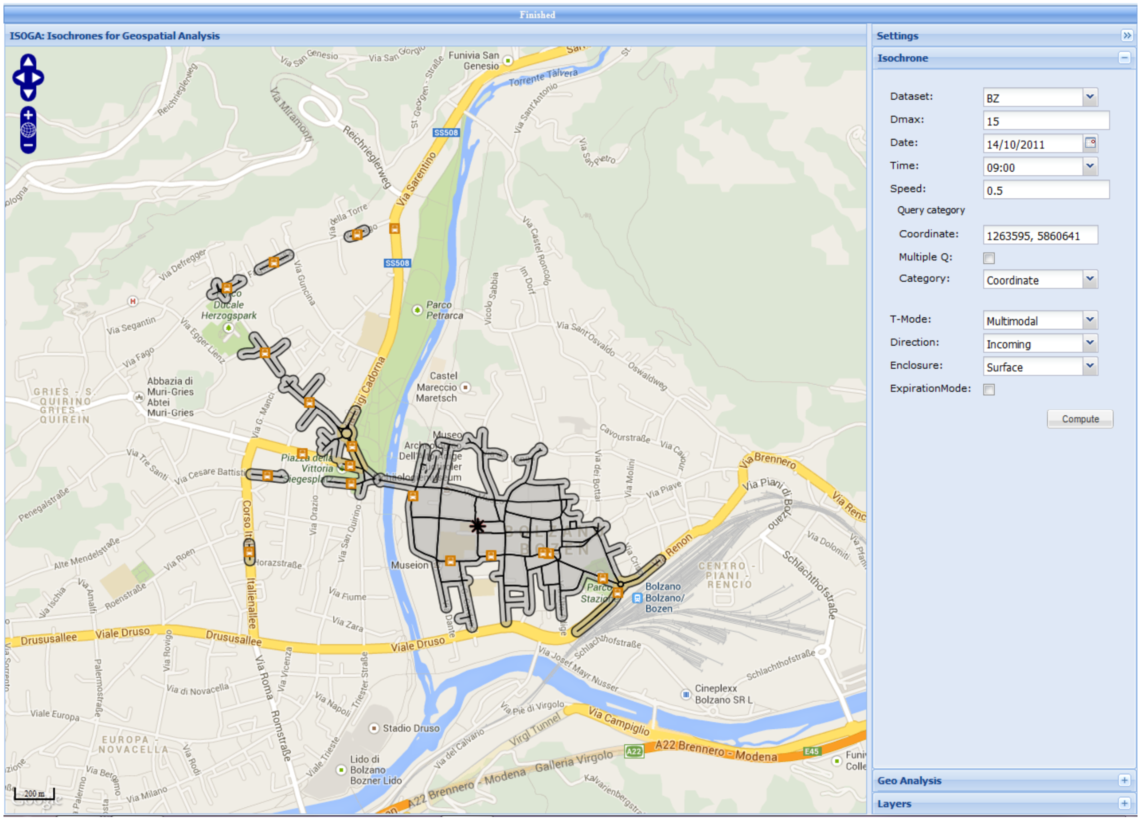
There were mainly two reasons that let me choose this project for my BSc thesis: Firstly, it is Open Source, which according to the *Open Source Initiative's Open Source Definition*[3] [sic!] means that it can be freely distributed as long as it retains its original Open Source license and is given with the application's source code. Secondly, it was made by someone within the Free University of Bozen-Bolzano, so that I could easily reach competent persons for help.

The target of the original isochrone project was to provide end-users a way to compute which parts they can reach from a given point on a map (since the Google Maps API is used to compute isochrones the available maps can be expanded by using other maps as long the necessary information like bus routes and stations are given) by only using public transportation systems (buses in this case) and walking. This is, for example, extremely useful for people who know that they are at some point at a given time and want to know if they can reach another point in a given interval, without effectively trying it by hard. This was done as a web project, so that the involved person could get the information at any given time as long as he/she has internet access, without the need of installing any proprietary program. Writing it as a web application allows the developer also to easily update needed information, such as changes in the time table on-the-fly without even being noticed by the user. Even more, it ensures that every end-user has always the latest version of the application, which is critical for such purposes, as the user does not want to get the time of a bus connection, which does no longer exist.

2.1 Functions of ISOGA

Although there is more functionality in the original project, such as executing SQL-commands based on the returned isochrone, here I will limit the description of the existing project to the basic elements.

[3] Open Source Definition available at http://opensource.org/docs/osd


Finished

ISOGA: Isochrones for Geospatial Analysis

Settings

Isochrone

Dataset: BZ
Dmax: 15
Date: 14/10/2011
Time: 09:00
Speed: 0.5
Query category
Coordinate: 1263595, 5860641
Multiple Q: ☐
Category: Coordinate
T-Mode: Multimodal
Direction: Incoming
Enclosure: Surface
ExpirationMode: ☐

Figure 2.1: Isochrone application (before changes were made) running on a local host

In the easiest way possible, the user simply drags the black point, from which the isochrone calculation will start, to its desired starting point and clicks the “Compute”-button. Then, the program starts to calculate the isochrone, zooms the map section so that the whole isochrone graph can be seen and draws it finally.

In reality in nearly all cases the user will not be satisfied by only referring to the given standard values and wants to input own parameters. While some parameters like “**Algorithm**” (chosen algorithm to calculate the isochrone; either MineX, MrneX or MDijkstra), “**Query Category**” (parameters for queries based on the drawn isochrone), “**T-Mode**”, “**Enclosure**” (how the isochrone will be displayed; either with surface filled or a buffer), and “**ExpirationMode**”, are mostly used by users interested in the technical aspect of isochrones, some other are interesting for every possible user These are:

- **Dataset**: the user is able to choose between a map of Bozen-Bolzano [BZ] South-Tyrol [ST], Italy [IT] or San Francisco [SF].
- **Dmax**: the user can choose the length of the maximal time interval of the isochrone.
- **Date/Time**: the user can choose the day and the hour for which he/she wants to calculate the isochrone; this can be especially useful for planning holiday trips, since then there are typically different schedules for the public transportation system than on regular working days.
- **Speed**: the user can alter its walking speed used for the calculation.

- **Direction**: the user can choose the direction, i.e., whether he/she wants, to arrive at the given query point or to start from the query point.

Besides these functions provided in the base version of the application, in this theses new functionalities were added. More specifically, I added the parameters **Time interval**, **interval for the size of the isochrone**, maximal t**ime** and i**sochrone size**, which are described in detail more detail in chapter three.

2.2 Technical Overview

As mentioned before, the isochrone program was written as web application, so that it is easily accessible from everywhere, provided an internet connection and web browser. This also means that – like most complex web applications – it consists of a 3-tier-architecture:

- An interactive **HTML page**, which enables the user to input the parameters through JavaScript (JS) and its extension Ext JS.
- A **Java application** running on the web server, which calculates the isochrone and the appropriate graphical representation on the map.
- A **PostgreSQL database, which stores the** data that is necessary to compute isochrones.

All three parts of the application can also be locally run on the own computer, so that any code changes do not directly influence the "official" web representation. Locally in this case is not the same as offline, since it still needs a connection to the internet to grab the necessary maps of Google Maps.

Basically we have the following workflow every time the application is used: When the end-user accesses the web application, the interface to put in all needed parameters and a standard section of the map is shown. If the user now moves the cursor-point (corresponds to the arrival or starting point of the isochrone request) and presses the Compute-button, the application (i.e., the Java application in the background) connects to the PostgreSQL database, tries to get the required data and computes the isochrone either once or multiple times. The progress of the whole calculation is shown as a status notification. When calculation is finished, the program sends an acknowledgement and draws the isochrone on the map. If more than one isochrone were computed, it will automatically show the first one.
If the user has computed at least one time in its current session an isochrone, he or she can afterwards also easily compute a new isochrone by simply dragging one of the two sliders, where one corresponds to the starting/arrival time and the other to the isochrone size (the time in minutes a transport can maximally take). The program will then automatically calculate the wished isochrone and show it.

Like in the previous chapter, I will only shortly describe different technical aspects and not go too much into detail as I used only a small portion of them in my own extension.

As a pure HTML web page would be too static, the application uses in its web interface heavily the JavaScript-extension **Ext JS**. JavaScript alone is used in nearly all modern homepages to make interaction with the user possible. Ext JS furthermore is an own JavaScript library which by inserting it into the project allows for even more customization and easy GUI building. This can be seen for example in the menu of the program, where the different elements correspond to different Ext JS forms like text labels, text and date fields, sliders or combo boxes.

For transporting asynchronous messages (e.g., subscribe the isochrone service and wait for the request) between the web server and the client, the application additionally uses the Bayeux protocol implemented into the **cometD** library. This allows for low latency requests.

When the request is then delivered the program asks for the necessary geospatial data from the corresponding **GeoServer**. This open source server then processes the data it gets from the corresponding **PostgreSQL** database server. The PostgreSQL database itself includes the **PostGIS** extension, so that geographic objects could be stored.

Although not directly linked with the execution of the program, it should also be mentioned that **Apache Maven** – a build tool – is used for automating the compile and build process for developers of the project. To compile a project (like the described one here in this thesis document) in Linux, many different libraries are required. Often it is difficult and very time-consuming to install all the libraries from which a project is dependent manually. The Maven (abbreviated also mvn) build tool describes these dependencies and allows for the automatic installation (and if necessary download) of them.

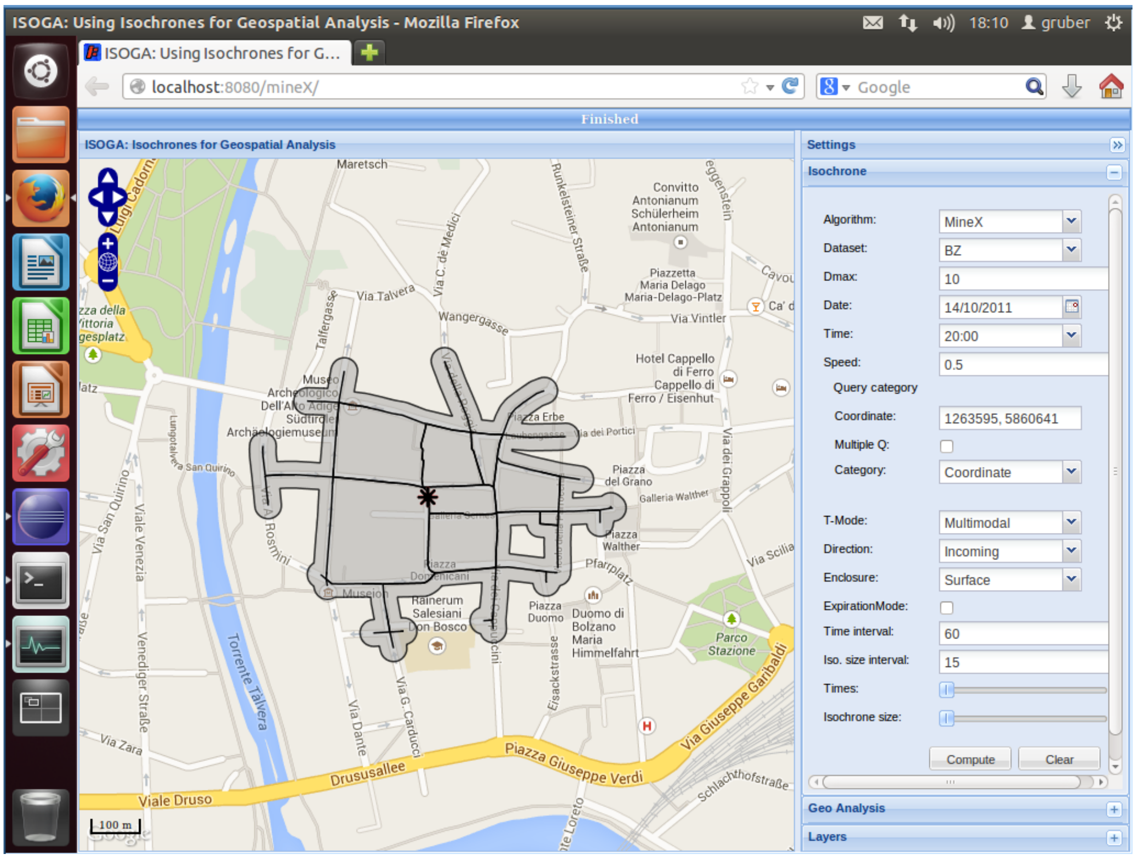
2.2.1 Algorithm to Compute Isochrones

There are three different algorithms to compute isochrones, which are included in the ISOGA application. They are called the MineX algorithm, the MrneX algorithm and the MDijkstra algorithm.

- The **MDijkstra** variant uses – as its name assumes – the Dijkstra algorithm (a BestFirstSearch variation), where immediate results are stored in RAM.
- The **MineX** variant is an "upgrade" of the MDijkstra algorithm. The main difference to MDijkstra is that it is a disk-based algorithm, which requires only a small amount of RAM.
- The **MrneX** is a further "evolution" of the previous algorithms. Instead of reading individual junctions, each database access retrieves a range-query. By doing so, the number of database accesses could be significantly reduced.

3. A Slider Mechanism to Browse Isochrones

After the brief summary of what the original ISOGA project consists of and a short analysis of some of its technical backgrounds, I will now focus on the expansions that I implemented during my thesis project. In particular, I will describe all the implemented features and how they were implemented. This will be done together with some effective code snippets to further make them understandable.



ISOGA: Using Isochrones for Geospatial Analysis - Mozilla Firefox
18:10
gruber
ISOGA: Using Isochrones for G...
localhost:8080/mineX/
Google
Finished
ISOGA: Isochrones for Geospatial Analysis
Settings
Isochrone
Compute
Clear
Geo Analysis
Layers
100 m

Algorithm:	MineX
Dataset:	BZ
Dmax:	10
Date:	14/10/2011
Time:	20:00
Speed:	0.5
Query category	
Coordinate:	1263595, 5860641
Multiple Q:	☐
Category:	Coordinate
T-Mode:	Multimodal
Direction:	Incoming
Enclosure:	Surface
ExpirationMode:	☐
Time interval:	60
Iso. size interval:	15
Times:	
Isochrone size:	

Figure 3.1: The isochrone application with the newly added expansions running on a local host

3.1 Computing Multiple Isochrones at once

The first extension concerns the computation of multiple isochrones at once. We did not develop a new algorithm from scratch, but re-used one of the existing isochrone algorithms to compute several isochrones in an iterative manner.

With the possibility of running multiple isochrone calculations one after another without additional user interaction, it is also possible to store the results of the computed isochrones.

With this information it is then in a later step doable to not recompute it, but directly gain access to the result and getting in this way the isochrones faster.

In practice this is done by letting the user initially insert some additional starting parameters. The main values used for this computation are:

- **Time**: the starting/arrival time of the first isochrone to be computed.
- **Dmax**: the isochrone size in minutes to be computed.
- **Time interval**: the interval between the different starting/arrival time of the isochrones.

After the user has inserted the required data (or does not change the initial standard values) and pressed the "Compute"-button, the method first checks how many isochrones need to be computed. This is done by transforming the starting/arrival time into a numerical value and then testing how many times it is possible to add the "time interval"-value before reaching the numerical value for midnight-time. The amount of needed isochrone calculations is then stored in an own variable. Midnight was chosen as a maximum to avoid any double calculations or any changes in the date.

The next step in the process is to compute the multiple isochrones. The parameters for the first computed isochrone are the values initially given by the user; the next isochrones differ from the first one by the starting/arriving time, which logically will increment in each step by the user-defined value.

During the whole process, a small status bar will appear at the top of the screen, which informs the user about the already processed isochrones. After everything has been properly calculated, the user receives a "Finished!"-message in the same status bar and the first isochrone of all the computed ones will be drawn on the map.

If the user later on wants to check out the same isochrone with one of the (pre)calculated parameters, he or she can simply do this by moving the "Times"-slider.

Technically, this feature is implemented via the following code sample:

```
// subscribes an isochrone request
$.cometd.subscribe('/service/isochrones', function(message) {
 var start = Date.now();
 var result = jQuery.parseJSON(message.data);

 LOG_PARSING_TIME += Date.now() - start;

 if (result) {
  if (result.status == "processing" && firstInvocation) {
    var prctg = result.processed;
    progressBar.updateProgress(prctg, prctg * 100 + "%
processed");
   } else if (result.status == "inserting" && firstInvocation)
{
    progressBar.updateText("Inserting Isochrone Nr. " +
(currentATidx+1) + " of " + arrivalTimes.length);
   } else if (result.status == "coveraging" &&
firstInvocation) {
    progressBar.updateText("Coveraging Isochrone Nr. " + (currentATidx+1) + " of " + arrivalTimes.length);
  } else if (result.status == "finished" && firstInvocation) {

    results.push(result);
    if (currentATidx+1 == arrivalTimes.length) {
     progressBar.updateProgress(1, "Finished");
     currentATidx = 0;
     destroyDynamicLayers();
     createResultLayers(extractFormParameters(), results[currentATidx]);
     firstInvocation = false;

    }
    else {

     currentATidx = currentATidx + 1;
     firstInvocation = true;
     $.cometd.publish('/service/isochrones', extractFormParameters());
    }
  }
 }
 LOG_PARSING_TIME += Date.now() - start;
});

// publishes the query
$.cometd.publish('/service/isochrones', queryParams);
```

Basically, we first define the service we want to subscribe to. Here this is done by `$.cometd.subscribe ('/service/isochrones', function(message){..}` and a following description of the function to be executed. `/service/isochrones` is given as the unique identifier of this specific function, so that we can afterwards publish it. The service itself is defined on another code page, exactly the **IsochroneService.java** file.

For this short code snippet I made simple changes to an already existing service for computing single isochrones, so that I could reuse code.

At any state after the isochrone service is started, it has an assigned result parameter. Depending on this "result"-variable it is possible to execute different tasks like showing the user at which step the execution actually is.

As it is not easily possible to stop a CometD service, in case of the isochrone service, it is needed to assign a variable, which marks that a specific iteration has already finished. This variable then holds the information if something was already processed. It was realized via the simple binary parameter `firstInvocation`.

If the current invocation is effectively in its first run, it is always shown that it is in this exact step and also in an own status bar, which is displayed at the top of the screen:

`progressBar.updateText ("Coveraging Isochrone Nr. " + (currentATidx+1) + " of " + arrivalTimes.length);`

The `arrivalTimes.length` is hereby the number of isochrones to be computed and was stored when the slider with all different times was generated.

After the computation of a single isochrone has finished, the `result.status` of the service returns `finished`, and the single result is pushed into the results array. Afterwards there are two different ways to proceed, depending on whether it is the last isochrone to be computed or that additional isochrones have still to be computed.

- In the first case (*last isochrone in a multiple isochrone computation*), the status bar shows that the process has finished (`progressBar.updateProgress(1, "Finished");`) and that the first isochrone will be drawn on the screen by the following command: `createResultLayers(extractFormParameters(), results[currentATidx]).`
- In the second case (*additional isochrones have to be computed*), the current index increases by one (`currentATidx = currentATidx + 1;`) and the computation for the next isochrone is requested by calling `$.cometd.publish('/service/isochrones', extractFormParameters()).` This step is repeated as long as there are still additional isochrones to be computed.

After the isochrone service has been defined the service has to be published and consequently be requested for the first time by the above mentioned `$.cometd.publish` -command.

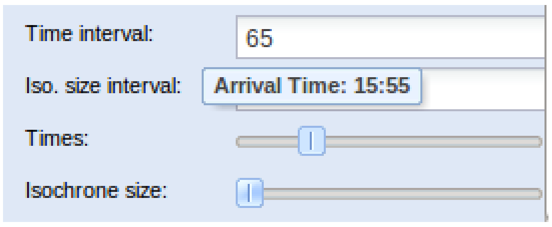
3.2 Slider Mechanism

The second extension to make the ISOGA application more user-friendly was to implement a simple slider mechanism. These sliders allow the user to change input parameters easily by only moving a slider to the left or to the right. Otherwise he or she would have to retype the parameters and "click" the "Compute"-button. While this sounds not like much, it still decrements the amount of clicks and movements needed in the long term.

In the program itself it was implemented by adding two new Ext JS slider elements and additional parameters that can be given by the user. Effectively used for the sliders were at one hand all the results given by the previous computation of multiple isochrones and at the other hand the "**Iso. size interval**"-text box-value. This parameter determines how much the isochrone size will change if the user alters the slider by one unit.

The two sliders are defined as following:

- The "**Times**:"- slider determines the starting/arrival time of the isochrone to be computed. Hence, it takes practically the same role as the "**Time**:"-text field, only in a more comfortable way for the user.


Time interval: 65

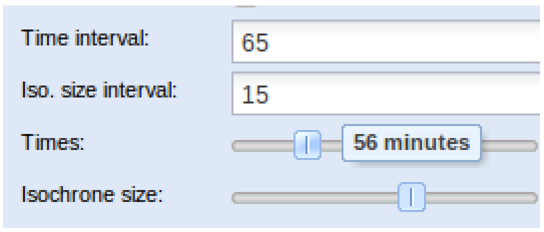
Iso. size interval: Arrival Time: 15:55

Times:

Isochrone size:

Figure 3.2: „Times“ slider with corresponding Tip

- The “**Isochrone size**:”-slider determines the size of the isochrone to be computed. As before, it takes the role of an already existing parameter, in this case the “**Dmax**:”-text box.


Time interval: 65

Iso. size interval: 15

Times: 56 minutes

Isochrone size:

Figure 3.3: „Isochrone Size“ slider with corresponding Tip

Both sliders will automatically show their current values if the user clicks on one of it (and probably moves it). Technically, this was done by a simple code variant of the standard slider tip.

```
var tip2 = new Ext.slider.Tip({
 getText: function(thumb) {
   var dmax = queryPanel.find('name', ID_DMAX)[0].getValue();
   var isoSizeInterval = queryPanel.find('name',
ID_ISO_SIZE_INTERVAL)[0].getValue();
   return String.format('<b>{0} minutes<b>', dmax +
(thumb.value)*isoSizeInterval);
 }
});
```

This code is then linked by

```
 plugins : tip2,
```

in the Ext JS – slider item definition.

The sliders take as input all the times computed in the last step described in chapter 3.1. Additionally, it also considers the values from both sliders themselves. If the user changes one of the sliders, the execution of the program can follow in two different ways:

- *Case 1*: the first slider is in any possible legal position and the second one remains at the starting position. This means that the user wants exactly one of the isochrones that were already computed. In this case, the program gains the result of the computation from the temporarily stored array and goes directly to the step of drawing the desired isochrone.
- *Case 2*: the first slider is in any possible legal position and the second one is not at its starting position. In this case, the user wants a slightly altered isochrone computation in comparison to the ones previously computed. The application will then directly compute that single isochrone and show it to the user.

Technically, the slider (as an Ext JS element) was implemented as follows:

```
queryPanel.items.items[0].add({
 fieldLabel : 'Isochrone size',
 xtype : 'slider',
 name : ID_ISOCHRONE_SIZE,
 id : ID_ISOCHRONE_SIZE,
 width : 130,
 increment : 1,
 minValue : 0,
 maxValue : 5
 plugins : tip2,
 listeners : {
  dragend : {
   fn:function() {
    var queryParams = extractFormParameters();
    var localDmax = queryPanel.find('name', ID_DMAX)[0].getValue();
    var isoSizeInterval = queryPanel.find('name', ID_ISO_SIZE_INTERVAL)[0].getValue();
    queryParams.dmax = localDmax + queryPanel.find('name', ID_ISOCHRONE_SIZE)[0].getValue() * isoSizeInterval;
    currentATidx = queryPanel.find('name', ID_ARRIVAL_TIMES)[0].getValue();
    computeIsochrone(queryParams);

   }
  }
 }
});

queryPanel.items.items[0].doLayout();
```

"`queryPanel.items.items[0].add` adds a new item to the Ext JS built menu (or panel) on the right of the screen. This item is defined by the content written in the following curved brackets, whereas each element has the following role:

The `fieldLabel` defines the text (label) written to the left of the menu item. Although it is not relevant from a technical side, it is useful for the user since it allows logically assigning a parameter to some specific functions.

The `xtype` defines the type of the Ext JS item that is needed. Some of the types needed in the program were:

- "**combo**" for a list of elements (as with the algorithm or Dataset field);
- "**numberfield**"/"**datefield**" and "**timefield**" respectively to show a text field with a limited amount of possible content (in this case only numbers, dates or times);
- "**checkbox**";
- "**slider**" (as used here).

The `name` and the `id` define a unique identifier of the specific item. This is essentially useful when exactly this item is needed later on. Afterwards it is possible to gain access to it by a simple name reference (e.g., `queryPanel.find ('name', ID_ISOCHRONE_SIZE).`

The value `width` defines the width [sic!] of the element in pixels. Sometimes this information is not needed as specific elements like a check box have no width.

The values `minValue` and `maxValue` define the minimal and the maximal internal value of the element's value. Besides that the value `increment` shows the amount of alteration of this value for one movement to the left or to the right. In our specific example, we can see that the slider has allowed (internal) values from zero to five and that one change increases or decreases the value by 1. The value shown to the user in the tip can be totally independent from it, and can be freely assigned in the code.

The `plugins` parameter allows for a further customization of the given element. In this case by a custom tip (appears when the slider is clicked) which was already described before.

While the previous code fragments describe the layout/design of the item, the real functional part is written inside the `listeners` part. In this section, it is possible to assign different listener[4] to the item and give them different meanings. After the "listener" is aware that the specific event happened (in this case the `dragend`[5]), the specified function will be executed.

Firstly, most needed parameters for a isochrone computation are directly read out of the form by the `extractFormParameters()` – function. Some others instead have to be read explicitly and shortly computed like the dmax[6]-parameter, which is combined by the base dmax value and the isochrone base size multiplied by the internal isochrone size interval value.

After all information for the computation was gained, the calculation is started by the `computeIsochrone(queryParams)` function.

4 An **event listener** catches when an event happens and starts then specified commands

5 The **dragend** listener is activated when the movement (the "drag") of the slider ends

6 The **dmax** defines the isochrone size in minutes (already mentioned in chapter 3.2.1).

After the item is defined, the layout of the page has to be refreshed. This is done by the `doLayout()`-command.

While this code example shows effectively the code of the “Isochrone size”-slider, it does not differ that much from the “Times:”-slider. The main difference to the “Times:”-slider is the function inside the listener-element:

```
listeners : {
 dragend : {
  fn:function() {
   currentATidx = queryPanel.find('name',
ID_ARRIVAL_TIMES)[0].getValue();
   destroyDynamicLayers();
   if (queryPanel.find('name',
ID_ISOCHRONE_SIZE)[0].getValue() == 0)
    createResultLayers (extractFormParameters(),
results[currentATidx]);
   else {
    var queryParams = extractFormParameters();
    var localDmax = queryPanel.find('name',
ID_DMAX)[0].getValue();
    queryParams.dmax = (queryPanel.find ('name',
ID_ISOCHRONE_SIZE)[0].getValue()+1)*localDmax;
    currentATidx = queryPanel.find('name',
ID_ARRIVAL_TIMES)[0].getValue();
    computeIsochrone(queryParams);
   }
  }
```

The big difference to the previous example is that this slider function includes a check if the isochrone was already computed before or not. This can be the case if exactly the wanted one was calculated during the computation of multiple isochrones. If this is indeed the case, the client has directly access to this isochrone and can immediately start to render it.

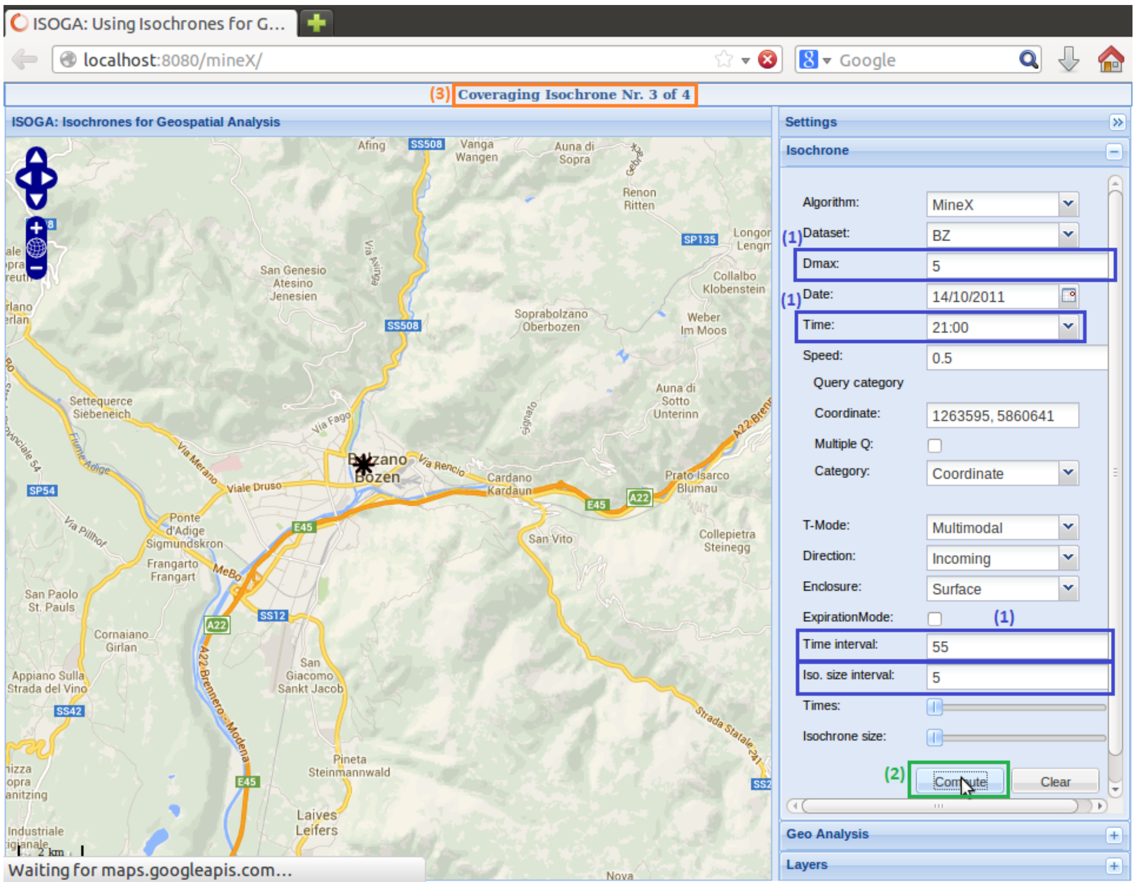
Technically, this is done by first checking if the other slider’s internal value is zero, which is equivalent to its starting value. This is realized by accessing a simple if-condition:
`if (queryPanel.find('name', ID_ISOCHRONE_SIZE)[0].getValue() == 0)`

If [sic!] the result is positive, the application will instantly call the method`createResultLayers (extractFormParameters(), results[currentATidx])` with the previously computed result and parameters, which will draw the isochrone layer without intermediate steps.

If this is not the case, all needed parameters are read, and the isochrone has to be computed before it is automatically drawn. This is precisely the same as within the first slider.

3.3 Running Example

To better illustrate the flow of events in the application, the following short chapter shows a running example.


ISOGA: Using Isochrones for G...

localhost:8080/mineX/

(3) Coveraging Isochrone Nr. 3 of 4

ISOGA: Isochrones for Geospatial Analysis

Settings

Isochrone

Algorithm:	MineX
(1) Dataset:	BZ
Dmax:	5
(1) Date:	14/10/2011
Time:	21:00
Speed:	0.5
Query category	
Coordinate:	1263595, 5860641
Multiple Q:	☐
Category:	Coordinate
T-Mode:	Multimodal
Direction:	Incoming
Enclosure:	Surface
ExpirationMode:	☐ (1)
Time interval:	55
Iso. size interval:	5
Times:	
Isochrone size:	

Waiting for maps.googleapis.com...

Figure 3.4: the application currently computing isochrones

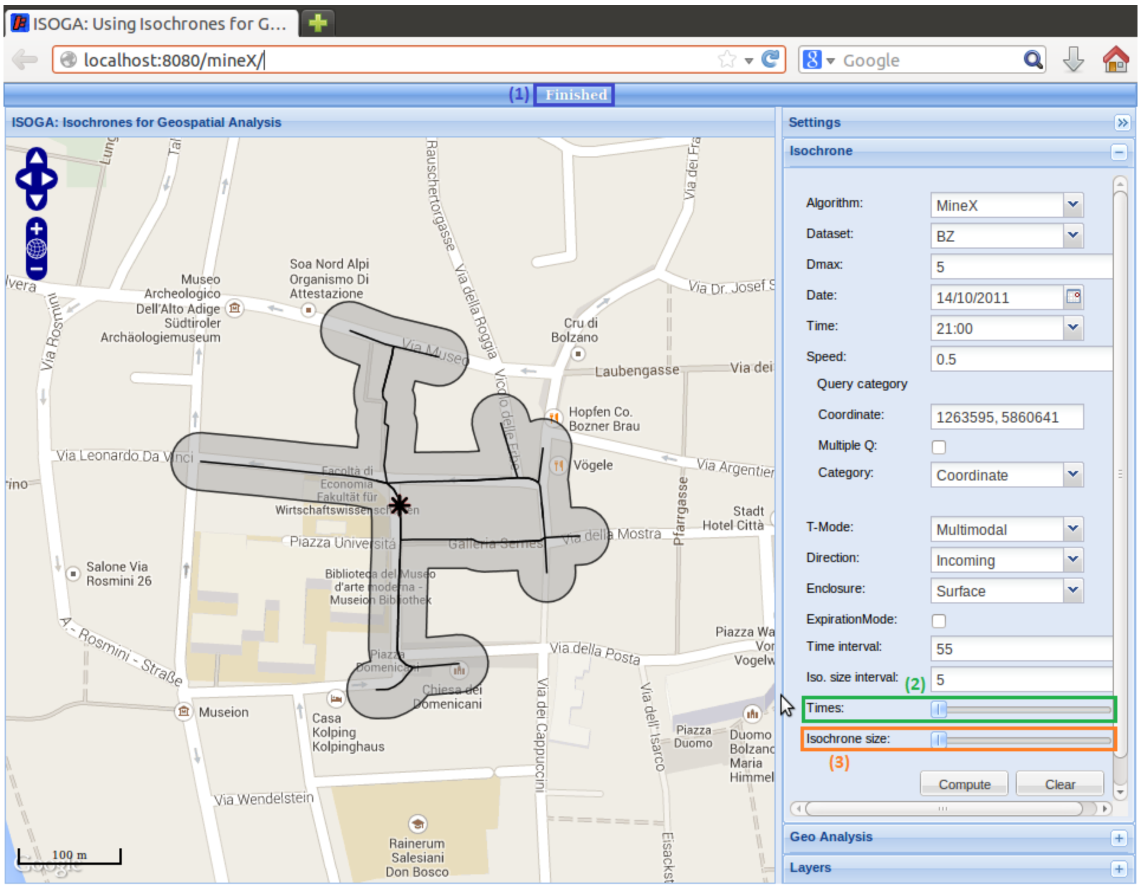
First of all, we need some starting parameters. While we could simply leave all values at their starting position, we change some of them to show differences. Although every point on the right “side bar” of the program is modifiable, it is only needed to adjust some of the most important of them for a basic test run. These are (marked by (1) in the figure):

• **Dmax**:	Set to 5 (minutes)
• **Time**:	Set to 21:00
• **Time interval**:	Set to 55 (minutes)
• **Iso. (isochrone) size interval**:	Set to 5 (minutes)

Besides that, it is worth mentioning that before the first computation the two sliders have no specified function because they have at that moment no data to work with.

After we put in the specified values, there are two ways to compute multiple isochrones. Either we drag the "star" on the map to our point of interest from which we want the isochrone to start or leave it where it is and simply press on the "Compute"-button (marked by (2) in the figure).

In both cases we will notice that a status bar will appear at the top of our browser window, showing some information of the current status of the operation (marked by (3) in the figure). First of all, it will show at which point of the current isochrone computation, but also at which isochrone number from how many at all it is. In our case when the first time was set to be 21:00 and the time interval between the different isochrone starting times was set to be 55 minutes it will compute 4 isochrones, namely 21:00, 21:55, 22:50 and 23:45. To avoid endless computation, it will end when the time of midnight is reached. In some cases this will also mean that only one isochrone will be computed at all (e.g. time: 23:00 and time interval: 65 minutes).



Algorithm: MineX
Dataset: BZ
Dmax: 5
Date: 14/10/2011
Time: 21:00
Speed: 0.5
Query category
Coordinate: 1263595, 5860641
Multiple Q:
Category: Coordinate
T-Mode: Multimodal
Direction: Incoming
Enclosure: Surface
ExpirationMode:
Time interval: 55
Iso. size interval: 5
Times:
Isochrone size:

Figure 3.5: the application has finished its computation

After the computation has finished (shown logically by a "Finished" in the very same status bar; marked by (1) in the second figure) the first computed isochrone will be drawn on the map. Now also the sliders have a function.

The first slider (the “Times:” slider; marked by (2) in the second figure) allows to change the starting time of an isochrone in an easy way by simply dragging it. If the isochrone size has still the starting value it will read the data from the previously computed step and show it directly on the map without computing it once more.

If we also move the second slider (the “Isochrone size:”-slider, marked by (3) in the second figure), we can as easily change the size of the isochrone. This is done by those steps that we previously described with the “Iso. size interval”-parameter (in our case 5 minutes): 5, 10, 15, 20, 25 and 30 minutes. A movement of this slider will let the application compute only the specifically needed isochrone and draw them afterwards.

If it would be needed to insert new values, we could simply press “Compute” or drag the “star” again, and it would overwrite the old computed values.

4. Future Possibilities and Changes

When someone works with an existing project (in this case the ISOGA project) and wants to improve it, it is only logical to think about what can be further done to expand it. This chapter will deal with this question and give proposals of what can be done in the future. These points can then be further elaborated and probably enhanced to a separate project.

Nevertheless, first of all I would like to mention that of course the ISOGA project has also many portions that are – according to my opinion – already implemented in a very good way and do not need much further changes. These are, for example, the *functionality* (there are no main parameters which aren't adjustable) or the *presentation* (all functions are easily accessible within a graphical user interface). But this chapter instead will focus on these parts that can still be improved.

Two points that – in my opinion - are quite interesting for future work are:

- a better reference to *mobile devices* (mainly smartphones);
- *a rebuild of the code* and the user interface and to make it *slimmer.*

As it is possible to see from the figures of the application, it is already fulfilled with possible parameters and user changeable values. This on one hand gives the user a wide range of possibilities to use the program exactly like he or she wants to. On the other hand, however, it also blows up the interface, so that not even all options can be seen on one screen. This effect gets even worse on mobile devices like smaller smartphones, were the screen is smaller and the interactive elements have to be displayed bigger to be easily touchable. These devices are of a special importance as many people would probably like to access route planning software not only at home, but at any possible location with an internet connection. A possible solution could be to only show a small amount of base parameters for the user and then to optionally allow the user to expand the list of extended features. In this case a redesign of the user interface would also be easier since not all parameters would have to be placed in a single window.

As smartphones gain more and more market share[7], it could also be important to revise the used **algorithm and to adapt it to mobile devices**. While the currently used variants all either store temporary results in a devices' memory (e.g. the MDijkstra algorithm) or disk (e.g. the MineX algorithm), it could be a possible improvement to store these on a web server. This would drastically reduce the foot print on the device but also lead to further costs as probably there would now be a need for (always available) web servers.

An improvement which would not be only beneficial for the user of mobile devices would be the implementation of an **internal help system**. As the application at the moment has so many different functionalities, it is often not clearly visible, which parameter has which effect. This problem could, for example, be handled by a helping function, which would give a short explanation for the different parameters. The main work that has to be done here would consist of the planning of how to implement it so that it does not overfill the already "overfilled" application.

[7] According to a survey from Bitkom in 2012 every third person in germany has a smartphone. [→http://www.bitkom.org/de/presse/64026_71854.aspx]

While the previous statements focused on “outer” aspects of the application, a possible future work could also be **a rewriting of the existing code** or, in an even more abrupt way, a complete new application with the same (or similar) features but a better structured code. This would have to be done because the existing code already got that big that it is not easy doable to add further changes. To arrange the code and put it up in a better structured way could even be more work than a complete rework because to do the former the programmer has to have insight into every little detail of the code and possible side effects.

Another possible improvement in the technique field would be a modernization of the used libraries. Ext JS is heavily used in the application, but in it’s now outdated version 3. The 4th version has included new functionalities especially for touch devices, which would be therefore predestined for an adaption for mobile devices. While this sounds like a slightly easier improvement, it is not. Through the circumstance that **Ext JS 4** has much new functionality it is not backwards compatible with Ext JS 3. So it would have to be evaluated if the work of (possible) rewriting of many code fragments is the new functions worth.

5. Conclusion

In this thesis I introduced a way to compute isochrones in a more interactive and user-friendly way than it was possible before. We begun by briefly introducing the concept of isochrones ("a line drawn on a map connecting points at which something occurs at the same time") as well as providing short overview of the ISOGA project, which uses isochrones to conduct geospatial analyses. Then, we introduced to extensions to the existing ISOGA application. First, a slider mechanism was added to the program, so that it is now possible to alter specific values for the computation of isochrones by a simple movement of a slider instead of a reinsertion of values and bugging repetitions of known steps. This permitted users to start the computation of similar isochrones swifter. Second, the possibility to calculate multiple isochrones at once was introduced. With this, the user (in specific cases) no longer needs to wait for the computation of the desired isochrone, but instead can be directly linked to the temporarily stored previous results. This allowed rendering the graphical representation of isochrones in a much faster way.

In a further attempt to ease the user's way not only to work but also to code with the ISOGA application, the appendix lists explains and lists all programs that were needed for the development.

While the main target of the thesis was achieved (i.e., making the application easier for the user to interact with), the concept could very well be further expanded to make the program even more user-friendly. Some directions for future work were mentioned in the chapter of future possibilities and changes. Some other extensions cannot be recognized today through the everlasting (r)evolution of technologies. A program may never be without flaws, but each improvement let us go nearer to that perfect status we want to achieve.

References

[1] M.Innerebner, and M.H. Boehlen, and J. Gamper. ISOGA: a System for Geographical Reachability Analysis Enhanced with Statistics. In Proc. of W2GIS-132, pages 9, April 3-5, 2013, Banff, Alberta, Canada.

[2] J. Gamper, and M.H. Boehlen, and M.Innerebner. Scalable Computation of Isochrones with Network Expiration. In Proc. of SSDBM-12, pages 526-543, June 25-27, 2012, Chania, Crete, Greece.

[3] J. Gamper, and M.H. Boehlen, and W. Cometti and M.Innerebner. Defining Isochrones in Multimodal Spatial Networks. In Proc. of CIKM-11, pages 2381-2384, October 24-28, 2011, Glasgow, Scotland.

[4] V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, and I. Timko. Computing isochrones in multi-modal, schedule-based transport networks (Demo paper). In Proc. of ACMGIS-08, pages 1-2, November 5-7, 2008, Irvine, CA, USA.

[5] S. Marciuska and J. Gamper. The Allocation of Dynamic Objects Within an Isochrone. In Proc. of ADBIS-10, pages 392-405, September 20-24, 2010, Novi Sad,Serbia.

[6] M.Innerebner, M.H. Boehlen, and I. Timko. A web-enabled extension of a spatio-temporal DBMS. In .ACMGIS-07, pages 34-41., November 7-9, 2007, Seattle, Washington, USA.

[7] Alexandros Efentakis, Nikos Grivas, George Lamprianidis, Georg Magenschab and Dieter Pfoser: Isochrones, Traffic and DEMOgraphics. (Demo paper) SIGSPATIAL/GIS 2013.

Appendix – How to Compile the Project

The biggest (initial) problem in working with the ISOGA project was not the code changes themselves but to make the application run. To avoid (or at least reduce) such obstacles for future contributors, I would like to add a small explanation of how to compile and start it. The text itself was mainly taken from the project's "readme" text file and was supplemented with further explanations by my own.

First of all to develop and work with this project, specific tools and/or libraries are needed, without them it is not possible to even start the application locally. The necessary ones are

- Linux (tested with Ubuntu 12.04 LTS[8])
- Java SDK[9] (at least version 7 or higher)
- Apache Maven (mvn)
- GeoServer
- PostgreSQL data base with PostGIS

After all these programs are installed they need to be configured and to be "feed" with the correct data. The PostgreSQL database and the GeoServer instance have to be arranged according to the **config.xml** – file found in the *resources* folder inside the *main* and *test* directories, so that the project can successfully access them. As a second option it would also be possible to alter the config.xml – file, so that it would match with the databases' data. After these configurations were adapted, the database has to be filled with the data from the following 3 SQL dumps:

- **bz_export.sql**
- **it_export.sql**
- **sf_export.sql**

Also, an additional SQL-command must be executed inside the database, which will alter or respectively add the **droptables** function.

```
-- Function: droptables(text, text)

-- DROP FUNCTION droptables(text, text);

CREATE OR REPLACE FUNCTION droptables(_schema text,
_parttionbase text)
 RETURNS void AS
$BODY$
DECLARE
```

8 **Ubuntu** can be downloaded from http://www.ubuntu.com

9 The current **Java SDK** can be downloaded from http://www.oracle.com/technetwork/java/

```
    row     record;
BEGIN
    FOR row IN
        SELECT
            table_schema,
            table_name
        FROM
            information_schema.tables
        WHERE
            table_type = 'BASE TABLE'
        AND
            table_schema = _schema
        AND
            table_name ILIKE (_parttionbase || '%')
    LOOP
        EXECUTE 'DROP TABLE ' ||
quote_ident(row.table_schema) || '.'
|| quote_ident(row.table_name);
        RAISE INFO 'Dropped table: %',
quote_ident(row.table_schema) ||
'.' || quote_ident(row.table_name);
    END LOOP;
END;
$BODY$
 LANGUAGE plpgsql VOLATILE
 COST 100;
ALTER FUNCTION droptables(text, text)
 OWNER TO postgres;
```

With this step, the PostgreSQL database should be properly arranged. The GeoServer instance will still need additional styles and workspaces. These can be found in the **geoserver_config** directory, whereby the content of the styles folder goes to GeoServer's **data_dir/styles** and the content of the workspaces folder to **data_dir/workspaces**.

After everything is properly constructed, the project can be compiled and run by executing the following commands inside the Linux terminal in the project's main folder or respectively the folder in which the **pom.xml** file, which is necessary for the Maven tool, is stored:

- **mvn clean install**
- **mvn jetty:run**

The application can then be accessed via a web browser[10]. If it was, for example, locally installed, it is located at **http://localhost:8080/mineX/**[11]

[10] It was tested by **Mozilla Firefox**, which can be downloaded from http://mozilla.org/firefox
[11] The number in the address can change if there was another port than 8080 used

Referenced and Used Software

- Official **Ubuntu** home page: http://www.ubuntu.com
  (Linux derivate on which the Isochrone project was tested)

- Official **Java** home page: http://www.oracle.com/technetwork/java/
  (necessary programming language to make own code changes)

- Official **Ext JS** home page: http://www.sencha.com/products/extjs/
  (JavaScript extension extensively used in this project)

- Offical **cometD** project home page: http://cometd.org/
  (implementation of the Bayeux-protocol in Java)

- Official **GeoServer** home page: http://geoserver.org/
  (an Open Source server for geospatial data)

- Official **Apache Maven** home page: http://maven.apache.org/
  (a build tool for Java software)

- Official **Mozilla Firefox** home page: http://mozilla.org/firefox
  (web browser with which the isochrone project was tested)