BigGeo: Visualizing large amounts of geo-referenced data

Frederico Santos
Instituto Superior Técnico
Lisbon, Portugal
frederico.b.santos@tecnico.ulisboa.pt

ABSTRACT

Smartphones have become part of our everyday life. These powerful devices allow us to perform all kinds of tasks, and are constantly saving data about where we have been, also know as spatiotemporal data.

The amount of spatiotemporal information, more precisely trajectories, created, and stored increased dramatically, and we are now in a Big Data era. To visualize this amount of trajectories, they cannot directly be drawn on a map. If we only draw one trajectory, it is easy to analyze it, but when that number turns to thousands, the screen gets heavily cluttered. Trajectories also have different attributes, such as speed or distance, which cannot be disregarded. The techniques that we will develop and apply also need to be user-friendly so that non-experts can retrieve valuable knowledge.

We propose BigGeo, a visualization which uses an interactive lens metaphor of different types, that allows us to filter the tracks and select which attributes to analyze. This visualization was evaluated through usability and scalability tests, which proved it was both user-friendly and scalable, meaning the objectives were met.

KEYWORDS

Gps, Smartphones, Big Data, Trajectories, Interactive lenses, Visualization, BigGeo

1 INTRODUCTION

With the rise of smartphones, computing power became accessible to anyone anywhere. There are now more than 3300 million of them worldwide, and what was only achievable in big computers in the past is now an easy task for the regular everyday objects that we carry everywhere in our pockets. We now have at our disposal millions of books and encyclopedias, stores and restaurants, social networks that let us connect with anyone on the planet.

This new chapter of human history, where we have such powerful tools at our disposal for almost 24 hours per day, means that we are generating more and more data every minute. Every day a staggering amount of data is generated, around 2.5 quintillion bytes.

This amount of data leads to problems regarding its storage and analysis, and the numbers do not seem to be slowing down. The term Big Data is everywhere, and it is essential to process it and analyze it because it allows us to make decisions promptly in order to save money and fix important issues [13], provided the necessary analytical process of transforming the raw data into high-level information is supported [4].

A significant portion of all data is Geospatial information, and in this field, there are several subgroups, including spatio-temporal data, which deals with positions attached to timestamps, such as trajectories.

Trajectories are an unusual type of data since they can be used to understand where and when someone has been or to study the significant patterns of trajectories within cities. So anyone can study this information, visualization tools need to be created. However, having to deal with a Big Data context brings several challenges on how to present and interact with it.

Visualization of any data has several problems being scalability and dynamics the two major challenges in visual analytics [11], and trajectories are no exception. If, for example, there is only one, it can be drawn on a map using a line that passes through every point of it, and we can understand where it passed through. The major obstacle is that this type of visual encoding works well for small numbers of trajectories (a few hundred) but not several thousands of them. If we draw a line for every single one on the map, the screen will get heavily cluttered.

When dealing with trajectory data, we have to take into account space and time. If we merely drew a line on the map, we would not be able to understand when that happened or how frequently we pass through it. The same principle applies to time, where if we list the trajectories in a timetable, we cannot understand where those passed through.

Designing such sophisticated tools for data visualization is a double-edged sword. On one side, we have the amount and type of information, and on the other, we have the complexity of the interface used to interact with it. Trying to make a user-friendly visualization is challenging since techniques that have better usability do not tend to scale well.

Objective

Given the problem mentioned above, our objective is to **design a map-based visualization of large numbers of trajectories, that allows their efficient and effective analysis**.

We want to be able to **visualize several thousand trajectories efficiently** and to **perform spatial and time analysis** while keeping the map's context. Attributes must not be disregarded, so we need to be able to **display several attributes at the same time**. These interactions **must be user-friendly** so that everyone can use it.

To meet these objectives, we must first have a preprocessing phase, which reads the GPX files, removes erroneous points, smooths out the trajectory, and to reduces the number of points necessary due to the scale of the problem, and leads to an accurate representation of the trajectory.

Several client-side algorithms will have to be designed as well in order to scale to a Big Data level effectively.

To meet these objectives, we developed BigGeo, which through the use of an interactive lens metaphor, lets users filter what trajectories they want or select which attribute is being studied, depending on the zone and features chosen, e.g., the time of recording or length.

To achieve this goal, we studied the current approaches followed to study trajectory data. Developed a web application, and performed user and scalability tests to determine whether or not our objectives were met.

Organization of the Document

We start by presenting the state of the art on trajectories visualization in Section 2. In Section 3, we describe the system's architecture, and the tools used to preprocess the raw data. Then details on how we implemented the visualization part and our interactive lenses work. In Section 4 we describe the tests performed, its results, and the statistical analysis performed on the results. Section 5 we reflect on the challenges found, whether or not our objectives were met, and plan for future iterations.

2 RELATED WORK

Here we will present state of the art in the field of Visual Analytics (VA), more precisely VA related to movement data, followed by the objectives we want to achieve with our study.

We found several studies using sparse datasets of trajectories, e.g., traffic cameras, meaning the path taken was unknown, consequently allowed for more smaller analysis, such as understanding how certain zones are affected by traffic throughout the day [12]. These tools provided us methods retrieve visual information from **incomplete trajectories**.

The second type of data we found on this subject consisted of social media posts, and it was analyzed to extract movement patterns [2]. The user would select places of interest, and then movement information between those, such as time or amount of people went from one to another, would then be shown in a circular bar-plot around the areas selected.

Complete trajectories

When dealing with complete datasets, we have to take other approaches.

Density fields were used in two boat trajectories visualization tools [8] [7]. This technique draws density fields around the trajectories' paths. These fields' radius and colors are then determined by the variables being studied. These had two problems. The first was that applying them in a city context would occlude the smaller streets. The second was concerned with the function system used to select the data, as it required a real analyst to create functions for an operator [7].

We analyzed a series of prototypes, where the technique **edge-bundling** was used among others [10]. The approach groups together similar data, trajectories that start and end close to each other and follow the same routes, in a single line. Since fewer lines are being displayed, the visualization tends to be more comfortable and less cluttered.

Trajectory Wall is a hybrid 3D and 2D visualization of a trajectory or set of them (passing through the same places), which places them on top of each (3D) other, ordered by time on a map (2D). The trajectory is divided into a set of bands that are either calculated or set by the user, and their color encodes the attribute being studied,e.g., speed, where red encodes low speeds and green big ones [9]. Although this approach is suitable for comparing a small number of trajectories, it can be hard to get valuable knowledge from many since the screen will get filled with bands. Another problem with using 3D is occlusion, but the visualization allows for rotation and zoom, to decrease the impact.

A similar approached studied the "crowdedness" of Tokyo's metro systems [5] through the height of the trajectory wall. Here the authors of the study analyzed the entry and exit stations where each person passes their card and calculated the path taken. Other than this, the visualization also tries to predict the path in case of accidents or significant events, for example, earthquakes.

We found a study [1] that used the magnifying glass metaphor. Here the users were presented with a lens that distorted the map below to get a closer view of a particular area. While the inner part of the lens is focusing on the small area, the rest of the map remains unchanged so that context is kept, this is called a **fisheye effect**. This type of approach where distortion of the map occurs is not something we were

interested in following since the data would get distorted as well.

Interactive Lenses

An exciting approach used the interactive lens, both as an analyzing tool and as a set of filters [6]. Their lenses have three types: origin, destination, and waypoint. These lenses are applied on top of the 2D map, and the thicker the lines they produce, the more trajectories pass through them.

The origin lens, filter only trajectories starting in the area of the lens. The destination follows the same principle where it only displays trajectories ending there, and the waypoint ones show trajectories that pass through them. Every lens can be placed in an area to find information about the number of vehicles in there or the number of trips, distance, and speed.

Using several waypoint lenses, the user creates several groups of trajectories, which is a very intuitive way of querying several groups, but the information displayed is not sufficient for a proper study of the data.

Discussion

The works presented before have several problems ranging from not being able to show significant amounts of data, not showing full trajectories or time, not displaying more than one attribute, or not providing a simple interaction.

The first feature we want to have is **scalability**. Our study deals with a big data problem, and so, having methods that display thousands of trajectories is extremely important. The first group of works were not able to represent a complete trajectory. On the one hand, they either handled start and end positions or sparse information, such as every 200 meters [12]. On the other hand, they could handle thousands of trajectories. The trajectory wall approach is not suitable for showing a significant number of distinct trajectories, because most get occluded. Density maps are also good at analyzing big datasets, but using them in a city context where roads constrain trajectories, and some of them quite small, with a lot of overlapping streets, does not work. Clustering can be a useful technique, but unfortunately, the algorithms take too much processing user dependable data. So filtering techniques will be used to scale to big data effectively.

We want to be able to show a **complete trajectory**, meaning techniques that apply for sparse datasets, are not suitable.

We want to perform temporal analysis on the data, since **time** is a necessity when dealing with trajectories, a successive collection of timed points on a map. We found that time lenses are a useful tool for filtering a portion of the trajectories that pass through them (necessary to analyze in a big data context), but other than this, no other visualizations could perform proper temporal analysis.

Trajectories are also attractive due to their **attributes**, so being able to show different ones such as distance, speed, duration, and length is essential. Most of the studies we analyzed were able to show at least one attribute. Techniques as density fields and trajectory walls are good at keeping the spacial context with several attributes.

Last but not least, we want our system to be used by non-experts. **Usability** is crucial for us since it will make the knowledge extraction more accessible, and we found that no techniques other than the use of interactive lenses [6] and trajectory walls were user-friendly.

This small summary comes to show that tools that can show both information on small and big datasets are rare, but when factors such as time, complete trajectories, and usability become essential, they are nonexistent. The previous tools are a view of what exists in the field of trajectory visualization, and they have exciting techniques to deal with our problem. Although they have flaws in certain key aspects, they are the stepping stone of our work. In our approach, we want to perform spatial and time analysis, be able to show several attributes from thousands of complete trajectories while remaining easy to use. To do so, and since we believe the current tools are not fit to deal with our problem, we propose BigGeo.

3 BIGGEO

Bellow we will explain our system and its features.

Our solution ocnsists of using a set of interactive lenses to filter the data, and a second one to display attributes as a visual property, e.g., velocity through color.

In order to achieve this, we needed to be able to preprocess our data. At this stage, erroneous positions and a simplification of the data was required to have a cleaner and efficient dataset, and it was achieved using an external library from a previously presented work [10], as this was not the aim of our study. We then developed further scripts to transform the gpx files in txt, and to eliminate some errors, such as repeated records.

Our tracks are then placed in a PostGIS database, and their geometries are calculated and stored. When storing the trajectories, we split them into four tables, according to their total length.

Overall we developed a web application using Node.js and Express as our server, together with a PostgreSQL database with the PostGIS features activated and Leaflet as our map-provider on the client-side.

To communicate between server and client, we use AJAX requests, containing properties of the lenses in the request (on client side), the server interprets these, creates a query and sends it to the database. The data is then sent to the client in GeoJSON messages, which is a form of JSON messages that contain geometries, and other attributes.

Lenses

With BigGeo, we reuse the interactive lens concept, presented in section 2, by having lenses that perform functions on trajectories that intersect them. These lenses are a way of "looking" more in-depth into the data.

We wanted a congruous object with which the user could interact with the data, and a simple concept so that it was easily understood. Taking inspiration on one of the already presented studies using interactive lenses [6], we wanted our lenses to be a way of filtering the data, but also a method of showing attributes inherent to each trajectory.

With our work, we introduce the concept of interactive lenses both as a filtering tool, an attribute selector, and an analyzer. Each lens is a circle that the user places on the map, which depending on its type, performs different actions on the trajectories that intersect it, the user can change their radius, move them and change their types.

Our system has two major groups of lenses, which we will explain in the following sections, the filtering lenses, responsible for selecting which trajectories are active, and the attribute lenses which represent a chosen attribute with one of four encodings, e.g., color.

Lens' basic dynamics The concepts we will present now apply to both types of lenses and consist of their **basic dynamics**.

When **placing a lens**, the user has to select one of their respective two buttons on the top side of the lateral menu. Then it must place the mouse on the position it wants to center the lens on, and it must drag the mouse while holding the left mouse button, until it has the right size. The radius is dynamically changed to the distance from the mouse to the position where the user started holding it (the lens' center). Upon the release of the mouse button, the lens is created.

Every lens object has a menu, and both contain three options that are equal to the user in the interface, the delete, which removes them, move which allows us to change their center, and radius button, which lets us change the radius.

In both lenses, we decided to place the menu attached to the center and not around the outside of the lens because when a lens is covering the whole screen, it would not be able to interact with the menu without zooming out or moving the map. Since it is harder to tell to which lens the menu belongs to with this approach, we decided to paint the inside of each lens when it is currently activated.

Visualization elements

The visualization starts by showing a map, the uploaded trajectories, and a side-menu.

On the menu, users can create the two types of lenses from our solution. Here users can also open a panel containing several histograms for them to analyze the distributions of all attributes of every trajectory active (only those that meet the filtering lenses' criteria).

Other than this, users can upload their trajectories, and on the settings panel, change the underlying map to one of five presets (to show labels or to see better), and change the scale from local to global or vice-versa.

When the global scale is activated, the intervals used to calculate the colors to paint or which width to use, from the attribute lens, are calculated according to the maximum and minimum of the whole dataset (all the trajectories in the database). When the local scale is active, the maximum and minimum represent the limits of the currently active trajectories (those being selected by filter lenses).

Filtering Lenses

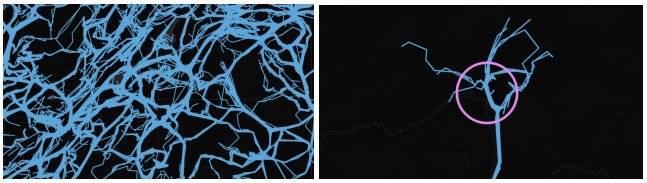

Figure 1: Left: Map without any lens; Right: Filtering lens of "End point" type applied, where the trajectories that end in it are activated;

The Filtering Lenses are, as the name suggests, a way of filtering the data, as can be seen in Figure 1 while going from the top image to the bottom one. When the user places a filtering lens X, which by default selects all the trajectories that pass through it, and then a second lens Y, which by default has the same behavior, he is selecting the trajectories that pass through both X and Y. The filtering lenses are mostly a set of dynamic filters. Dynamic because the user can change its radius, move it around, delete or even change what subtype it wants to apply, which translates into a different subset of tracks displayed.

When a filtering lens is created, a request is made to the server. Our server contains a string where it saves the conditions that each active filter lenses represents, which is sent to the database to retrieve our data. With every change to the position, radius, or subtype of any filter lens, a new request is made to our server that contains the previous state (old position, radius, and subtype), and the new one so that we can update the previously said string. One of the challenges of this approach was learning how to effectively communicate between client and server to make sure the results would not be incorrect due to the functions asynchronous nature.

Three types of lenses only select the tracks that either pass or start or end inside them. Then there four other types, which are essentially special conditions. Apart from passing

through the lens, each one is represented by a different type of filtering lens. There are seven different types:

- ***Pass by*** - This is the default type of lens. When the user places a lens on the map, he is creating a "Pass by lens" that is selecting all the trajectories that pass through its area.
- ***Start point*** - These lenses select all the trajectories whose starting point is inside the lenses' area.
- ***End point*** - These lenses select all the trajectories whose ending point is inside the lenses' area.
- ***Average velocity*** - These lenses select all the trajectories that both pass through its area and whose average velocity is inside its user-defined intervals whose extremes go from zero to the maximum stored in the database.
- ***Duration*** - Selects the trajectories that pass through its area and whose total duration is inside its user-defined intervals whose extremes go from zero minutes to the maximum size stored similar to the previous.
- ***Length*** - Selects the trajectories that both pass through its area and whose total length, in kilometers, is inside its parameterizable intervals, similar to the previous.
- ***Time*** - Selects the trajectories that both pass through its area and whose timestamp of creation (timestamp of the starting point) is inside its parameterizable intervals, similar to the previous.

The intervals of the last four lens types are parameterizable and can be changed through the lens menu either by sliders for length, duration or velocity, or a calendar selection for time.

Attribute Lenses

The previous lenses alone could not provide us with a way of really analyzing the overall dataset. To make sure we could visualize the features of each trajectory, we created the **attribute lenses**.

When placed above trajectories already drawn on the map, the attribute lenses access their attributes' value and change a visual characteristic of them to represent its attribute, e.g., the lenses see that the trajectory has a low-velocity value and paints it red.

Another difference between the two is that the data queried for is contained inside the lens. In the previous lenses, their queries selected trajectories that intersected them; these select the portions that are contained inside them, of those that are currently active. The tracks already queried by the filtering lenses do not change.

There are two aspects we need to understand for these lenses:

- ***Attribute*** - These are properties of the trajectories already calculated and stored in their respective rows in the database, such as average velocity, duration, or the total length.
- ***Visual encoding*** - A way for data to be translated into a visual structure that is shown on the screen. We wanted it to be associated with each of the trajectories themselves, so we decided to apply those changes to the visual properties of the layer, such as their color or opacity. When analyzing an attribute of a trajectory, we have several intervals established, whose ranges depend on whether the local or global scale is activated.

These lenses have different combinations depending on the attribute the user wants to study, and whichever visual encoding is selected. The user can change the type of attribute and visual encoding through one of two the top right side buttons of the menu.

Attributes. Here we will explain every attribute that can be selected with our system. The user can change them through the menu, and the possibilities are:

- **Velocity** - This value is read from one of the columns of each trajectory, which holds the average velocity of each point in the trajectory and represents its velocity in kilometers per hour.
- **Length** - The length attribute represents their size in kilometers and is calculated using one of PostGIS's functions, which returns the total length of the trajectory.
- **Duration** - The duration is calculated through the starting and ending points' timestamps, as we convert the time to UNIX, and calculate the difference.
- **Time** - The time attribute refers to how long ago the trajectory was created and has 24 different bins.
- **Day of the week** - Represents the days of the week of when it was created, which is calculated through the starting point's timestamp. Due to the nature of the attribute, there are seven different intervals, one for each hour.
- **Hour of the day** - Represents the hours of the day when it was created, which is calculated through the starting point's timestamp. Due to the nature of the attribute, there are 24 different intervals, one for each hour.

Visual Encodings. Now we will present which encodings we have, and the user can change them through the menu. The **Color** encoding refers to the layers' color, and its also the default type for every attribute lens created. The color goes from 00ff00 (green) to ff3232 (red), using a HSV gradient, as seen on the left lens of Figure 2, on every attribute. Depending on the attribute, we divide the color scale mentioned into as many intervals as it has (e.g., seven for days of the week and ten for length). This results in the same scale for

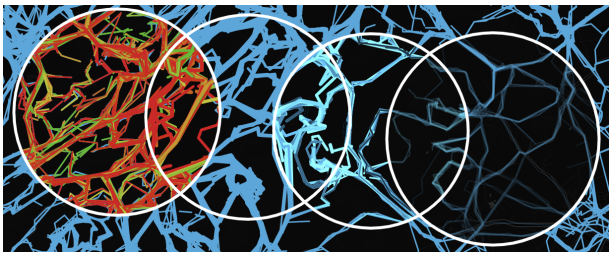

Figure 2: Example of four different attribute lenses with distinct encodings, starting with; color, width, brightness, and opacity. In the intersected areas, both lens' encodings are activated.

every attribute, attributed to the fact that it only differs in the number of steps, making it easier to quantify its values.

With every attribute combination, the color green represents the newest or highest value(highest UNIX time value), while red stands for the oldest or smallest values.

To apply the color to each trajectory, we go through the trajectories contained in the group layer of the lens and change their style.

When the user selects **color and time**, a different interaction occurs. Since we wanted to be able to study time better, we needed to understand how long ago the trajectories were recorded and to make sure this perspective could be reproduced, and we decided to add more colors to the gradient.

The second option is the **Width** encoding, which is the second lens from the left on Figure 2, starting at zero, and one being the default value. The width of each object goes from 1 to 5.5, and the space between each value is determined by the number of intervals in each attribute. We decided to set the maximum to 5.5 because when the width was above that, it was hard to keep the context of where the trajectory passed through.

A third possible option is the **Brightness** encoding, visible on the third lens from the left in Figure 2. The brightness of a color refers to its lightness or darkness. To calculate it, we have to take into account the existing color, and its interval, $[-1, 1]$, which represents black and white, respectively. The new color is then passed as the color property of the layer's style.

The **Opacity** encoding corresponds to the transparency value of a color and its another property of the style object of the GeoJSON layer (the object containing the trajectory) from Leaflet. It has the interval $[0, 1]$, where zero represents a fully transparent object while one means it is opaque and can be seen on the rightmost lens of Figure 2.

This encoding was a lot more challenging than the others, mainly because of how we organize the layers. With the attribute lens, we are only placing more layers on top of the already drawn ones. We do this because we did not want to change the underlying layer of the live tracks, due to the time it takes to draw. Since the underlying layer does not change, the opacity of the attribute lenses' layer (above) would not affect anything because the layer bellow was opaque (we would see through the first one but not the second, making it seem as if the opacity encoding did not work).

To counter that, we needed to keep track of the opacity lenses and update the layer bellow whenever some changed or was created by removing the intersected part.

With every attribute combination, higher widths, being more opaque, and brighter represent newer or higher values, while smaller widths, more transparent, and dark ones are older or smaller values.

Len's statistics. Each attribute lens possesses a circular barplot, which represents the number of trajectories contained in each interval, of the attribute being analyzed. It lets us analyze the distribution of the numbers of trajectories, contained in the lens, for each interval of the associated attribute. This can also be used to understand which colors represent which interval.

Attribute lenses combination

In this subsection, we present how the process of combining two or more attribute lenses works. Our attribute lens solution allows us to analyze the attributes, but that would be ineffective if we could not compare more than one at the same time.

When an attribute lens is placed on top of another, its layers are drawn on top of the others, meaning we cannot view the other's attributes. To counter this, we wanted the attribute lenses to combine their effects.

We could not create a layer for a trajectory and apply one style to half and another to the second half. To counter this, we needed to separate the trajectory, so we decided the lenses would not interact with the trajectories themselves. Instead, a new data object was created, the **areas**.

Each attribute lens can have more than one **area**. Each **area** has a different set of encodings applied, because **area** is a different set of intersections with other lenses. If lens B is placed and it intersects lens A, then lens B has the following areas; the area that does not intersect lens A, the area that does, whereas A does not change and only has one.

Areas

Each area is an object which contains a polygon created using Turf.js polygon creator, a reference to the lenses that affect it, and the ID of the layer group that contains the GeoJSON layers (which are the trajectories).

To query the tracks the portions of the tracks contained in them, we convert the polygon object to the GeoJSON using Leaflet and pass it in a request to the server. The server then

creates a query which is passed to the database. The query selects the intersect portions of the current trajectories and returns it.

When any change is made to the type or encodings of the lens, its areas update their styles accordingly.

To calculate the areas of intersection, we developed an algorithm that goes through the areas of the lens that intersect the newly placed lens and calculates their intersections.

Zoom Filtering

As we explained before, our trajectories are stored in four tables according to their length. In pursuance of less cluttering, we decided that not all data should be shown at all times.

At low zoom levels (zoomed out), most trajectories are displayed as a mere pixel, but almost none of the works presented before had any filtering according to their zoom.

When the client requests the server for a new set of active trajectories (due to changes in the filter lenses), it actually makes four different requests, one for each table, and saves each in a different layer. The layer containing the biggest trajectories is always active, no matter the zoom level, and all its trajectories have more than four kilometers. The remaining three layers are activated in descending order in the event that analyzes the changes to the zoom level and activates or deactivates them accordingly.

Other than filtering the trajectories by zoom, we also apply different simplifications in each of the four layers.

4 EVALUATION

The main objective of our evaluation was to understand if users could use our system to make the types of analysis we set out to do on section 2.

To perform said tests, we conducted a set of usability tests with a set of six tasks, and a questionnaire was applied containing the system usability scale (SUS) and the NASA-TLX test, among other feature-related questions.

The tasks were performed on our system using a public dataset containing 15k trajectories, mostly on foot by tourists, on the Balearic Islands, Spain.

Scalability tests were also made to determine if our system was, in fact, scalable. Here we determined to times it took to perform several actions on different sized datasets, going from 1k to 1M trajectories.

Usability tests

We developed a set of six tasks. These consisted of filtering the tracks that passed on predefined locations (through markers created) and performing some analysis. We took note of the times each participant took completing the task, the number of errors given, the difficulty, and the success rate (wrong answers and withdraws are considered equal here). The tasks are:

- State the number of trajectories that start in the lower marker in a four-kilometer radius and pass through the highest one, on a four-kilometer area as well.
- Of all the trajectories that pass in a radius of 4.5 kilometers from our marker, please state the biggest one.
- From the trajectories that pass through an area of nine kilometers around the marker, and whose duration is between one hour and an hour and a half. Please state which has the highest average velocity.
- From the trajectories that pass through an area of 5.5 kilometers around the marker, state which was the day of the week where most of them happened.
- From the trajectories that pass through an area of 2.4 kilometers around the marker, and whose start hour was between 20 and 24. Please state which of those was the fastest.
- From the trajectories that pass through an area of 500 meters around the marker, please state their average duration.

The first group of tasks is composed of the first and last tasks. These require the user to place the filters lens on top of the markers to select the trajectories desired trajectories. Then the users should open the statistics panel and retrieve the desired information. These are the most straightforward tasks and will let us know how well the users understood the basic concepts of our system and how they interacted with the lenses. With these, we can confirm that if users can, in fact, perform spatial and temporal analysis.

The second group is comprised of the second, third, and fourth. Here the users need to place the filtering lenses too and then must interpret those results, with the use of histograms, and give a concrete answer. These are harder than the previous group, although their complexity is not high.

The last group is harder than the previous two. It only has one task, number five, where the user must place a filtering lens and two attribute lenses, with different attributes and encodings, obviously. Other than being a way of evaluating our lens combination feature, it also has the purpose of confirming one of our objectives, being able to analyze several attributes at the same time. Being able to complete this task means that the user fully understood our concept.

Users

A total of 20 people volunteered for our tests. We performed each task once to each participant, followed by the questionnaires. Their age groups were, 80% between 18 and 25, 10% from 26 until 45 and 10% from 46 to 60, and none had any visual impairments that could cause them to perceive colors wrong.

We also decided to see if any had previous experience with either trajectory visualization or trajectory recording, and only 10% had done any of those, and none had performed both.

Results

Task 5 had a 45% success rate, and task 3 and 4 had 85% and 95% respectively, whereas the remaining had 100%.

The results we found seem to indicate that task number 5, was, in fact, the hardest according to the users. When studying the confidence intervals for the mean with a value of 95%, it shows that tasks 1,2,4, and 6 have their difficulty values between 1 and 2.5, whereas task 3's difficulty is between 3.06 and 4.04 and the fifth task's one is between 4.09 and 4.81. Task 5's difficulty mode was, in fact, 5, making it the hardest of them all.

When examining the other variables, we can argue that a future user would take from 3:45 to 4:39 minutes to complete the fifth task with a 95% confidence interval, which is pretty close to the limit given for the completion of the tasks, five minutes. Tasks numbered 1,2,4, and 6 had similar intervals, going from one to two minutes in duration, with the same confidence level of before. On average, people take less than two minutes completing tasks 1, 2, 4, and 6, but they take around 3:12 to finish task 3 and 4:34 to finish task 5.

Regarding the number of errors, tasks 3 and 5 also stand out, as we can claim with a 95% confidence that people will make between 2 and 6 errors in task 5 and 1.8 to 4.4 in task 3. By looking at these results, it is easy to state that tasks 3 and 5 are harder and that people will always take more time or make more mistakes in them, but we need to confirm that these differences are, in fact, statistically significant.

In order to affirm that the differences between the tasks are significant, we applied the Kruskal-Wallis H Test on the time, difficulty, and errors and the Chi-square test of independence on the success rate.

Regarding all aspects, the null hypothesis was tested and rejected, meaning there were significant differences. Given the rejection, the posthoc tests allowed us to conclude that task number 5 is significantly different from tasks number 1, 2, 4, and 6 on the success rate and the number of errors.

Regarding the difficulty and time taken, through the posthoc tests, we found there were two groups, tasks number 3 and 5 (first group) were significantly different from all other tasks (second group), but there was no difference between them.

In general, task number 5 was the hardest of them all, followed by task number 3, which fits with the difficulties assigned by the users.

Questionnaires results

We calculated the average results of our SUS related questions and used these to get the final result, where our application scored 74.25, a B regarding the industry's standards.

We decided to apply the raw NASA-TLX in the final questionnaire of the test instead of at each group of tasks because we wanted to find the overall value of our application. Here we scored 45.83. Although there is not an industry-standard over what value makes a good interface, we found a study [3] that evaluated that values between 40 and 50 were considered functional interfaces.

Other than these tests, we also asked users about how they felt about certain features individually. When asked how important the statistics panel or the circular bar-plot were, 95% of users said it was either a four or a five, being five "crucial" and one "unnecessary".

Then they allowed us to determine color encoding was the one most user understood better, whereas the width attribute was the worst.

Most users liked interacting with the lenses since when asked how they felt about moving, changing their radius and types, and the regarding the zoom filtering 50% of the users, on a scale of one to five, being five the "Really important" and one "Unimportant", voted four and 35% five.

Empirical evidence

Now we will present some observations on some of the tasks.

The fifth task's performance suffered a lot due to users not understanding that they could not filter for the hour of the day with a filter lens. This can be attributed to the existence of the "Time" filtering lens, which, as explained, most users would select. Since most users did not remember that a second attribute lens was possible to place, even though it was shown in the introduction of the tests, the success rate of this question was shallow, 45%.

Another problem associated with task number five was the use of the width encoding. When users selected it, they could not understand the difference between the values. The opacity encoding, together with color, were the two most used encodings. The use of the opacity lens is best applied to small sample sizes of trajectories because a high number of low opacity trajectories stacked results in an opaque line. This, together with the fact that most users used this encoding together with the color one, resulted in several wrong answers.

The other tasks performed well, as there were a lot of common points between all. Once users learned how to move and change the types, tasks would be a lot more fluid, by virtue of our system's low learning curve.

Overall of the feedback was really good as users really enjoyed testing it, and some even wanted to try it out further after the tests.

Scalability tests

In this brief subsection, we will explain another group of tests we applied to our system.

This second type of tests was aimed at our scalability objective. A series of tests, with different sized datasets, were used to determine the times their queries took as well as to find the time it took while drawing the queried tracks and changing their styles,e.g., when an attribute tracks is placed. We created seven distinct datasets with varying sizes, as stated previously, and we would load them into our database. Our datasets contained; 1k, 10k, 100k, 250k, 500k, 750k, 1M trajectories of random lengths.

We then proceeded to place filters lenses and removed them. The action of removing every filter lens requests all trajectories in the dataset, the worst-case scenario, which we took note of the several times, to query and draw. The times were calculated through the server's terminal and the browser's console.

To calculate the time it took to change the style of every trajectory, we removed the 25 kilometer limit for the lenses size, and placed one on top of every trajectory, and changed the encodings. The process of changing the encoding was measured through the browser's console.

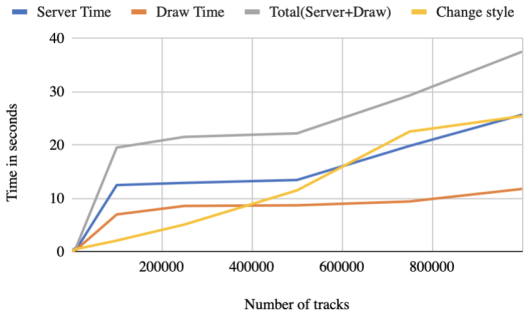

Figure 3: Time scalability according to the number of trajectories.

Obviously, the system took more time querying for one million tracks than any of the rest, which took 28.7 seconds, while the whole process took 37.5 seconds. With 750k trajectories and the server took 19.85 seconds to receive a result from the database, and the total was 29.3. As we can determine from Figure 3, our server time starts to increase a lot faster than our drawing procedures on the client-side.

The draw time remains rather constant as the datasets size increase from 100k up to 1M. Finally, the change style functions' time increases similarly with the server' one.

We can observe that our drawing procedures took a rather constant time, even when scaling to 1M tracks. This can be attributed to the use of a canvas renderer, the use of geometry objects, the simplification stage of the preprocessing phase, and the zoom filtering feature, which separates the trajectories according to their length.

Our back-end processing, related with fetching the requests, performing the respective query, waiting for the results, and sending them to the client-side, was not the aim of our study and took the worst performance, scaling almost linearly after 500k.

The time it took to change the styles of the already drawn tracks was slightly disappointing as it took a similar time as our server requests and queries.

Results discussion

The first objective, being able to visualize full trajectories, was present in every task. Since our system draws the trajectories in their complete form, and they are not calculated based on sparse point datasets. Other than this objective contains another supposition is being able to perform spatial analysis. All our tasks are related to performing some spatial analysis.

Being able to perform **time** analysis was our second objective. We can associate tasks numbered, 3, 4, 5, and 6 require some time analysis. Other than task 5, we were happy with the success rate of the tasks. Analyzing the NASA-TLX's results of performance, the users believed they were able to perform the necessary analysis, 76.50 (the highest of all other NASA-TLX categories), attributed to this we conclude this objective was met.

Our third objective was being able to visualize **several attributes** at the same time. Ascribed to this, we defined task 5. Its success rate was 45%, due to a number of observations already presented before, but we believe the one that impacted it the most was actually them not understanding that there was not a filter lens that could filter by the hour. As the question was written, it leads to the belief that they needed to filter by the area (a step all of the users completed) and then by the hours. Although the task was not as successful as the others, we believe this requirement was met.

Regarding the **scalability**, we performed the stress tests on our system. The results of these tests showed that our total time of creation (both on the server and drawing), followed a linear function. When we analyzed these, we found that the drawing time was almost constant, even when scaling to 1M, thanks to our discussed features implemented. Our Server time, on the other hand, took an inverse route as it scaled almost linearly after 500k. As this was not the aim of our study, we did not invest too much here. Our aim was to develop a visualization, which can later be connected to a better back-end to increase its potential further. Our change

style procedures were similar to the server's time, not the ideal results. Nevertheless, overall, we believe our results were satisfying. With this, we consider that our objective was met, although more work needs to be done.

Last but not least, for **usability**, we tested the raw NASA-TLX score on the overall of all tasks and the SUS test, as well as the other questions already mentioned about the specific features. Our system scored 74.25, a B in SUS, and 45.83 in the NASA-TLX, which places it in a good usability category. Although the results could be better, we need to take into consideration the reality of our problem, which is being able to perform analysis on big geo-referenced data. This adds an extra layer of complexity to our system that we had to overcome. Given the scores mentioned before and the nature of our context, we believe this objective was met.

5 CONCLUSIONS

We found a lot of trajectory data is being recorded, and that studying it allows us to solve several mobility related problems. Through our study of state of the art in mobility visual analytics, we found that there were no tools that could handle big trajectory data in a user-friendly way. So we set some objectives as to what to achieve.

We started by using preprocessing tools to clean, simplify, and transform GPX files int txt ones, allowing us to have a smaller and cleaner dataset. Then we uploaded these to a PostGIS database that connected to a Node.js web application.

We created a set of interactive lenses that allowed us to filter the trajectories needed and a second one that transcribed their attributes into visual encodings, such as the tracks' color. These allowed for myriad scenarios to be studied consistently.

We filtered the data using the zoom level, according to its length, reducing the overall clutter and increasing performance. We also created two scales that let users study homogeneous datasets.

We tested our solution with 20 participants. They answered questionnaires where we applied the raw NASA-TLX and the system's usability scale tests, which placed our system in good usability overall. These completed a set of tasks, while notes were taken on their times, errors, difficulty, and success rate, and later used to evaluate our system's usability. Then scalability tests were performed on our system to determine how it behaved when loading different amounts of trajectories.

We concluded that tasks 5 and 3 were harder than the remaining, but we were still happy with the results.

Through the scalability tests, we observed that our threshold should be around 250k tracks has of here, the system takes a long time to calculate changes, although our front-end could handle 1M tracks efficiently.

Concerning all of the above, we believe the objective of our study was achieved.

REFERENCES

[1] Caroline Appert, Olivier Chapuis, Emmanuel Pietriga, Caroline Appert, Olivier Chapuis, Emmanuel Pietriga, High-precision Magnification Lenses, and Caroline Appert. 2010. High-Precision Magnification Lenses To cite this version : HAL Id : inria-00474022 High-Precision Magnification Lenses. *SIGCHI conference on Human Factors in computing systems* (2010). https://hal.inria.fr/inria-00474022/document

[2] Siming Chen, Xiaoru Yuan, Zhenhuang Wang, Cong Guo, Jie Liang, Zuchao Wang, Xiaolong Luke Zhang, and Jiawan Zhang. 2016. Interactive Visual Discovering of Movement Patterns from Sparsely Sampled Geo-tagged Social Media Data. *IEEE Transactions on Visualization and Computer Graphics* 22, 1 (2016), 270–279. https://doi.org/10.1109/TVCG.2015.2467619

[3] Alexandra Fernandes and M. H. R. Eitrheim. 2016. The NASA Task Load Index for rating workload acceptability The NASA Task Load Index for rating workload acceptability. October (2016). https://doi.org/10.13140/RG.2.2.22989.64484

[4] Fosca Giannotti, Mirco Nanni, Dino Pedreschi, Fabio Pinelli, Chiara Renso, and Salvatore Rinzivillo. 2011. Unveiling the complexity of human mobility by querying and mining massive trajectory data. (2011). https://doi.org/10.1007/s00778-011-0244-8

[5] Masahiko Itoh, Daisaku Yokoyama, Masashi Toyoda, Yoshimitsu Tomita, Satoshi Kawamura, and Masaru Kitsuregawa. 2016. Visual Exploration of Changes in Passenger Flows and Tweets on Mega-City Metro Network. *IEEE Transactions on Big Data* 2, 1 (2016), 85–99. https://doi.org/10.1109/TBDATA.2016.2546301

[6] Robert Krüger, Dennis Thom, Michael Wörner, Harald Bosch, and Thomas Ertl. 2013. TrajectoryLenses - A set-based filtering and exploration technique for long-term trajectory data. *Computer Graphics Forum* 32, 3 PART4 (2013), 451–460. https://doi.org/10.1111/cgf.12132

[7] Roeland Scheepens, Niels Willems, Huub Van De Wetering, Gennady Andrienko, Natalia Andrienko, and Jarke J. Van Wijk. 2011. Composite density maps for multivariate trajectories. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (2011), 2518–2527. https://doi.org/10.1109/TVCG.2011.181

[8] Roeland Scheepens, Niels Willems, Huub Van De Wetering, and Jarke J. Van Wijk. 2011. Interactive visualization of multivariate trajectory data with density maps. *IEEE Pacific Visualization Symposium 2011, PacificVis 2011 - Proceedings* (2011), 147–154. https://doi.org/10.1109/PACIFICVIS.2011.5742384

[9] Citation Tominski, Christian Tominski, Heidrun Schumann, Gennady Andrienko, and Natalia Andrienko. 2012. City Research Online City , University of London Institutional Repository Stacking-Based Visualization of Trajectory Attribute Data. (2012).

[10] Daniel Ven. 2018. Understand My Steps ^. May (2018).

[11] Lidong Wang, Guanghui Wang, and Cheryl Ann Alexander. 2015. Big Data and Visualization : Methods , Challenges and Technology Progress. 1, 1 (2015), 33–38. https://doi.org/10.12691/dt-1-1-7

[12] Zuchao Wang, Tangzhi Ye, Min Lu, Xiaoru Yuan, Huamin Qu, Jacky Yuan, and Qianliang Wu. 2014. Visual exploration of sparse traffic trajectory data. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (2014), 1813–1822. https://doi.org/10.1109/TVCG.2014.2346746

[13] Arkady Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. 2011. Sensing as a Service and Big Data. (2011). https://arxiv.org/abs/1301.0159