TÉCNICO LISBOA

FastGeo

Visualizing large amounts of geospatial data in real-time

João Lucas Pereira Fidalgo Rafael

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Daniel Jorge Viegas Gonçalves
Prof. Daniel Filipe Martins Tavares Mendes

Examination Committee

Chairperson: Prof. José Luís Brinquete Borbinha
Supervisor: Prof. Daniel Jorge Viegas Gonçalves
Member of the Committee: Prof. João Manuel Brisson Lopes

December 2020

Acknowledgments

I would like to thank my supervisors, Daniel Gonçalves and Daniel Mendes, for sharing their knowledge and helping me improve and iterate over the many concepts and ideas presented in this thesis. Additionally, I would like to thank João Moreira for the insight and support throughout this year.

To my great friends, Beatriz Alves, Francisco Campaniço, Hugo Guerreiro, and Matilde Ramos. I would like to thank them for their presence throughout these paradoxically very long yet very short five years. They have helped me immensely, and I would not be the person I am today without them.

Finally, I thank my parents for their constant support, and all they have taught me in life. It is due to all their help that I was able to study something I have a deep interest in, and no amount of thanks is enough to acknowledge it properly.

This work was partially supported by FCT through project VisBig PTDC/CCI-CIF/28939/2017.

Abstract

The availability of devices that can record locations, as well as their constant connection to the Internet, creates a large amount of geospatial data that are continuously being streamed in and allows us to visualize this data in real-time, identifying important trends in traffic.

However, the quantity of data that is produced today makes this a challenge. Its processing has to be fast enough to handle a constant stream of new data. Apart from having to be visually clear, the visualization methods must also be displayed on-screen at interactive frame rates.

To study a way to show trajectory data in a way that is not only visually clean but efficient enough to handle large amounts of data constantly being streamed in, we conceived FastGeo, a way to visualize this data by displaying it in three different periods. We gradually simplify data from period to period, and in each using several representations, such as heatmaps and trajectory bundling, so that each period looks distinct and maintains good performance.

We developed a prototype to implement these concepts and tested it with users, and we concluded that they understood the way data was represented and how its trends changed over time. We evaluated this prototype's performance as well, which let us conclude that although the visual performance was effective, the scalability of trajectory bundling is limited and requires further improvement.

Keywords

Data Visualization, Visual Analytics, Trajectory Visualization, Geospatial Data.

Resumo

A disponibilidade de dispositivos que gravam localizações, tal como a sua constante conexão à Internet, cria uma grandes quantidade de dados geoespaciais que são recebidos constantemente, e permite-nos visualizar estes dados em tempo real, identificando tendências importantes no trânsito.

No entanto, a quantidade de dados produzidos hoje em dia torna isto um desafio. O processamento de dados tem de ser rápido o suficiente para lidar com um fluxo constante de dados novos. Os métodos de visualização alem de terem de ser visualmente claros, devem também ser mostrados no ecrã a *frame rates* interativos.

Para estudar uma forma de mostrar trajetórias de forma a não só serem visualmente claras, mas também eficiente o suficiente para lidar com grandes quantidades de dados a serem recebidos constantemente, desenvolvemos um protótipo que mostra dados em três períodos de tempo diferentes, ao mesmo tempo. Em cada período, usamos representações diferentes, tais como *heatmaps* e *trajectory bundling*, de forma a que cada período seja visualmente distinto, e para manter um bom desempenho.

Testámos este protótipo com utilizadores, e concluimos que eles conseguiram compreender como os dados são representados, e como as tendências nos mesmos mudavam ao longo do tempo. Também avaliámos o desempenho deste protótipo, o que nos levou a concluir que, apesar do desempenho visual ser bom, a escalabilidade do *trajectory bundling* é limitada e precisa de ser melhorada.

Palavras Chave

Visualização de Dados, Análise Visual, Visualização de Trajetórias, Dados Geoespaciais.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

With the advancements in computing that have happened over the past few decades, access to geographic data has become increasingly available. Everyone can know where they are in a few seconds, using modern GPS mechanisms that have errors in the order of fewer than ten meters. The assessability of this kind of data has led researchers to study it in the field of Big Data, which is an increasingly studied area in Computer Science, and the amount of geospatial data is by no means small: *"with the wide availability of GPS, RFID, remote sensor and satellite technologies, trajectory data can be collected on a massive scale and real-time manner"* (Deng et al., 2015). We have not only large amounts of data at our disposal, but this data is also produced in real-time.

However, having all this data means nothing if we cannot make sense of it. Therefore, we need to visualize it somehow. By visualizing this data, we can identify trends. Geospatial data can be divided into subgroups such as spatial events (combinations of locations and time intervals) or trajectories (sets of positions with associated timestamps). Trajectories can show where someone has been, and they may show patterns when in great numbers. Spatial patterns that show which streets tend to have the most traffic or what roads get more congested are important. However, when this trajectory data is being received in real-time, we can also focus on the temporal aspect: spatiotemporal patterns, such as what roads get less congested in the morning and what roads keep the same traffic when rush hour is over.

1.1 Problem

As with other types of data, Information Visualization allows exploring geographic data. However, geospatial data is especially complex. X and Y coordinates are complicated to analyze, of course, but

unlike most data, you cannot simply put geospatial data on a scatter plot. In this case, the coordinates correspond to real places in the world, and they are tied to timestamps that transform the coordinates from places into events. *"Visual Analytics can help by combining automated analysis with interactive visualization for effective understanding, reasoning and decision making on the basis of a very large and complex dataset"* (Gomes et al., 2017).

Therefore, we must find some way to visualize this data so that we find it easy to analyze and understand. When there is a massive amount of it, visualizing trajectory data is no small feat, though, and several challenges are posed regarding how to show and analyze it. The more constraints we put on this data (such as being a large amount of it), the more problems arise. Visualizing one trajectory is a matter of simply drawing it on a map, but were we to apply the same solution to larger amounts of data, it would be quite hard to understand what is shown due to too much visual clutter on-screen. This is the most important problem: we have to show large amounts of data, make it understandable, and avoid visual clutter, but ensure that we are not simplifying it too much.

Moreover, if we want to visualize the data in real-time and show the temporal side of trends and patterns as we discussed, more problems will arise, such as how to differentiate more recent data from older data or how to view recurring events and changes in routines over time. Simply drawing lines on a map and updating them with new information every few seconds will not tell us much about when the older lines happened or how frequently objects pass through the same places, so we must display much more information to make analysts understand what they are viewing.

1.2 Motivation

If we solve the problems that come with visualizing trajectories in large amounts and in real-time, we will end up with a visualization that can show spatiotemporal patterns, as we have discussed. If we can observe these patterns, we can study them and ask ourselves questions about why they happen.

Urban planners can study these patterns and try to identify roads that need more lanes due to being overly congested, and when they build these lanes, they can use this data to identify what streets and roads to use as detours to minimize the impact of construction on traffic. Advertisers can study the most congested streets during rush hour and focus on those streets when putting up billboards. If people can effectively visualize this data, then they can understand patterns and use this understanding for myriad goals.

1.3 Objective

Our work's primary objective is **to study a visually clear way to visualize large numbers of trajectory data as they are streamed in real-time, allowing the recognition of spatiotemporal patterns while the data is updated over time.**

To achieve this objective, we will have to clearly show time and space, enabling the user to analyze data without feeling overwhelmed by its quantity. In order to let users understand patterns in the data, it is also important that they have a good understanding of how old data is or how many objects passed through a given location, meaning we will have to show this as well.

Since we are handling large amounts of data, computational performance is a concern as well, and it is further amplified by this data being updated in real-time. Naturally, before data is displayed, it has to be processed (especially when developing different visualization techniques instead of showing it as-is), and this processing has to be fast enough to keep up with regular data updates.

Data will be updated gradually, resulting in its quantity getting bigger and bigger over time. To keep the amount of data from increasing uncontrollably, we will use a graceful degradation concept, which gradually simplifies data as it gets older, reducing the amount of data that needs to be dealt with in a controlled manner (Fialho and Gonçalves, 2018). This gradual simplification consists of dividing time into different periods and simplifying the data when it transitions from more recent periods into older ones.

1.4 Contributions

To achieve our objective, we tested different ways to visualize data with users to evaluate their usability. Additionally, we prototyped a system that would handle each visualization's data processing and simulate real-time data streaming using a static dataset. Our contributions are the following:

- Several visualization techniques that use trajectory bundling and grid binning to represent data.
- Time period concept that gracefully degrades geospatial data, transitioning it between each time period and representing each period with a different visualization technique.
- A prototype system to process trajectory data and prepare it for visualization, available (as well as the visualization techniques) on a GitHub repository [1].

[1] FastGeo: `https://github.com/lucasrafael98/FastGeo`, last visited December 30th, 2020

1.5 Document Structure

The following sections will be organized as such. Chapter 2 presents the related work we found in geospatial data visualization, both in real-time and for large amounts of it. In Chapter 3, we propose concepts regarding the visualization techniques and algorithms for data processing. Chapter 4 details how the concepts and algorithms in Chapter 3 have been implemented in a prototype and said prototype's architecture. In Chapter 5, we describe the usability and performance tests we performed, and then we analyze the results. Finally, in Chapter 6, we present our conclusions and discuss future work.

Chapter 2

Related Work

In this section, we will discuss the state of the art in Visual Analytics (VA) related to trajectory and movement data. We will start by introducing some basic concepts related to this area and then analyze and discuss several approaches to visualizing the aforementioned types of data.

The problem being tackled is visualizing geospatial trajectories, both with a large dataset and while additional trajectory data is being streamed in real-time. Therefore, we will focus on geospatial data visualization techniques related to these constraints when discussing related work. If we discuss any works that are not as closely related to this subject, they will nevertheless be discussed to understand the main focus better.

2.1 Basic Concepts

There are three fundamental types of spatio-temporal data (Andrienko et al., 2013). Spatial events refer to objects with limited time of existence, and they are a combination of a location in space and an interval in time and may have related thematic attributes. Time series are variations of thematic attributes over time, and they may or may not be related to a specific point in space. Trajectories are the most complex data type of the three (and this work's main focus), describing positions of moving objects at sampled time moments. Even though trajectories are the main focus of this work, both spatial events and time series will be displayed in the visualizations we will discuss.

Movement data must be preprocessed before visualization. A common problem with data measured in real life used to be that there was a small amount of it, and the intervals between measurements might be too large or irregular. For the former problem, the data might end up being unusable, but for the latter,

interpolation (estimating the spatial positions in regularly spaced time steps) can be used and needed for re-sampling (obtaining positions in regularly spaced time steps) (Andrienko et al., 2017).

Nowadays, in the age of Big Data, such problems are not as widespread, but data is still GPS (meaning there is a lot of associated error), and it is usually noisy as well. We can also compute attributes such as traveled distance, duration, and average speed from raw data. While trajectory data can be abstracted into origin-destination (OD) flows, which will distill a trajectory into a line between start and end points, this work focuses on viewing trajectories. Therefore, there must be an attempt not overly to rely on such strong simplifications of trajectory data.

There are many ways to visualize spatiotemporal data. While point-based visualizations are the most direct type of visualization for presenting and analyzing it, discrete sample points can be converted into continuous line forms that display paths from objects such as vehicles, humans, or marine life. Such line-based methods can also encode related variables with visual channels such as width or color hue, saturation or luminance, facilitating the depiction of spatial patterns (He et al., 2019).

He et al. (2019) discuss typical interaction techniques for trajectory data visual exploration into multiple categories. **View Manipulation** (e.g., panning, zoom, rotation, filtering, etc.) is used to select objects of interest and enables observation and analysis of trajectory scenes from various perspectives. **Data Manipulation** consists of selecting attribute information to represent values in a trajectory dataset. **Spatial Exchanges** display interregional exchanges on a map, but spatial distribution is lost in the process. **Temporal Mechanisms** may be several things: separating views such as sequential heatmaps may lose the user's focus, but time lenses can present temporal aggregation based on user-defined queries. **Map Interactions** such as choropleth maps (maps where distinct regions such as countries or states have encoded variables with visual channels such as color) enable better exploration of multivariate data (that is, data with several non-spatiotemporal attributes). **Edge Bundling** is good at reducing clutter and is the most popular solution to achieve this, though some challenging issues such as legibility, algorithmic complexity, and intuitive navigation may exist. **Focus + Context** enables a global contextual view and detailed focus views.

2.2 State of the Art

There are many works on the effective visualization of large scale geospatial data, each with its own approach to the problem. As for displaying this data as it changes in real-time, there are fewer works. Each subsection will analyze several works related to these problems.

2.2.1 Real-time and Streaming

As previously mentioned, there is little research related to visualizing spatiotemporal data in real-time. Gomes et al. (2017) developed a visualization that supports up to hundreds of thousands of data points at interactive framerates (20 frames per second). Each data point is animated in real-time and has color-coded trails with the data point's velocity (Figure 2.1). User controls include simple pause, play, rewind, and fast-forward functions for better data analysis.

Raw data input undergoes preprocessing, such as cleaning inaccurate, duplicate, or unrealistically high-speed points, precomputation of attributes such as distance and speed (so that it does not have to compute them afterward), and trajectory normalization (to ensure that all trajectories have the same time intervals between position updates). This normalization is used because the animations follow the pose-to-pose principle of animation, interpolating between the normalized keyframes to make it appear as if the objects are moving between them.

When necessary, the client-side sends its location to the server at regular intervals. To reduce the amount of communication between client and server, a dead-reckoning approach is used in which the server-side can estimate object position by using previous information, such as location, speed, and direction. A framework proposed to discover hot motion paths (Sacharidis et al., 2008) exploits the

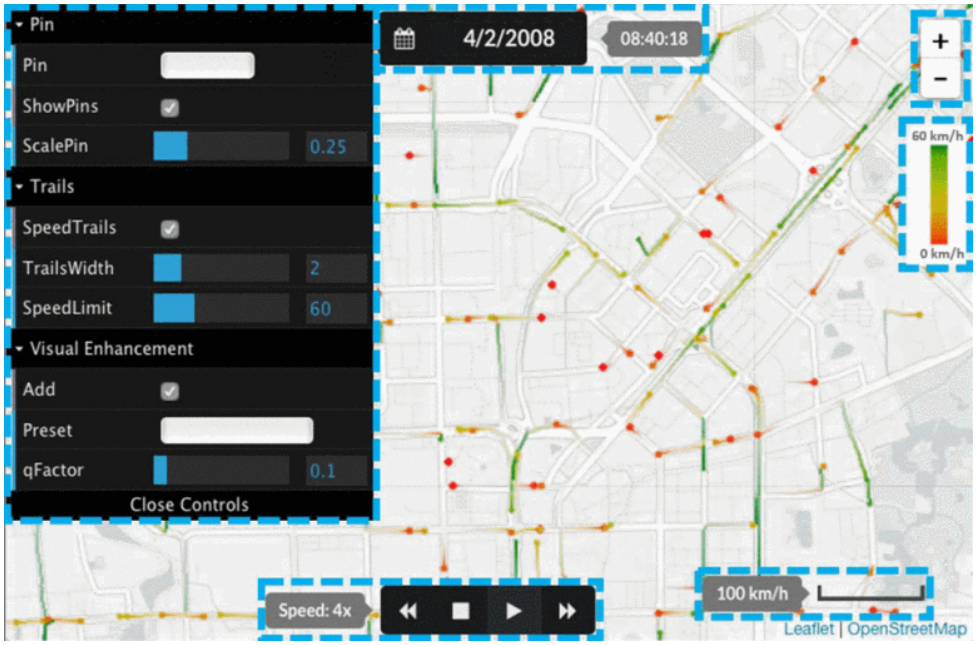


Figure 2.1: Traffic dynamics visualization from Gomes et al. (2017).

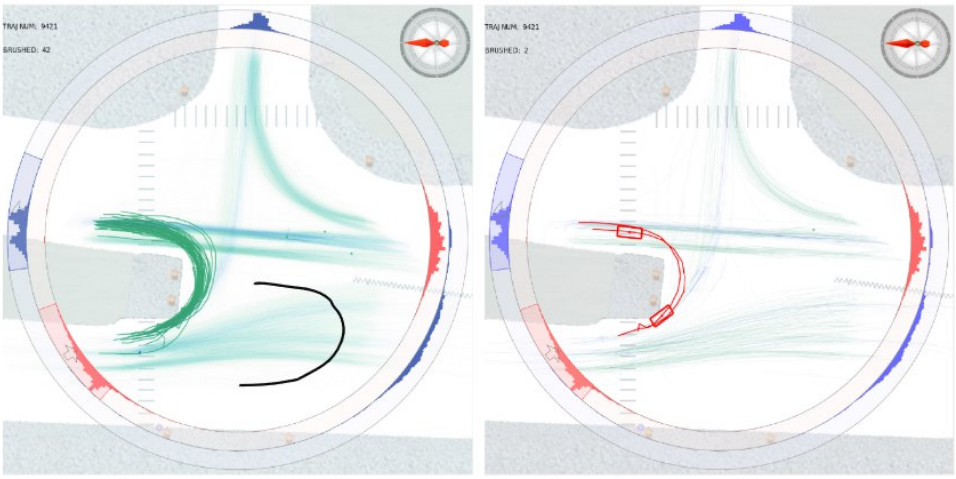


Figure 2.2: The user can draw a trajectory estimate, and TripVista (Guo et al., 2011) will display similar trajectories.

capabilities in the client-side to process and filter location updates to reduce the amount of processing required in the server-side and enabling this work to run receiving a streamed data source.

Each data point trail is rendered with GPU acceleration. Trail length and color depend on the point's velocity, and the trail fades away over time according to an exponential function. The trails are represented by texture-based primitives (such as lines and circles) instead of geometric, ensuring the required interactive framerates.

Filtering is a useful functionality in visualization, its main purpose being to allow the user to select only relevant data and ignore irrelevant parts. There are several ways to do this, but the easiest way for spatial filtering is to allow the user to draw one or more rectangles or circles on the map (Andrienko et al., 2013), and only objects that fit in or intersect with these simple shapes will remain visible after selection. For temporal filtering, the user can select a time interval from the data time span, and the visualization will only display trajectories and events that exist in the selected interval. Special attributes such as speed or traveled distance in a time interval may also be used to filter trajectory segments (for example, the user may want to remove very slow movement events such as being stuck in traffic).

TripVista (Guo et al., 2011) takes the concept of letting the user filter the visualization by drawing shapes, and combines this concept with the concept of a lens that encodes additional information overlayed on the map, which we will discuss in closer detail in the following section.

The spatial scale is smaller than most works we discuss in this section, focusing on a single inter-

section (Figure 2.3 (a)). Data was obtained with several laser scanners, and any object entering it was detected, tracked, and further classified as a car, bicycle, pedestrian, bus, or other. Naturally, this type of data is subject to noise and uncertainties caused by occlusions, so any trajectories with a very short passing distance or time were removed in preprocessing. However, there is an attribute shown by the lens view that is not displayed away from the main map in many other works: trajectory shape and direction. TripVista's lens shows the number of trails coming from and leaving each part of this intersection. Moreover, the user can draw rough estimates of a trajectory, and TripVista will display similar trajectories, as shown in Figure 2.2.

The user can also focus more on trajectories and attributes by selecting different attributes that can be encoded into the color hue of each trajectory, such as object type, that is, whether the trajectory belongs to a car, a bicycle, or a bus (Figure 2.3 (b)), and the speed of each trajectory (Figure 2.3 (c)). The aforementioned radial histogram displaying trajectory direction in the lens can also filter a particular trajectory type, such as trajectories starting from or ending on a certain part of the intersection.

A time slider can be used to select certain time intervals, and play and rewind buttons allow the user to watch movements unfold in real-time, with the position and shape of objects also represented on the map, at a minimum of twenty frames per second if middle-sized data is loaded. To mitigate some of the demand in computing resources when visualizing large datasets, intermediate results of all views are saved, and only the necessary parts are updated when small changes arise from user interactions.

The visualization is not solely composed of this intersection view. A ThemeRiver (Havre et al., 2002)

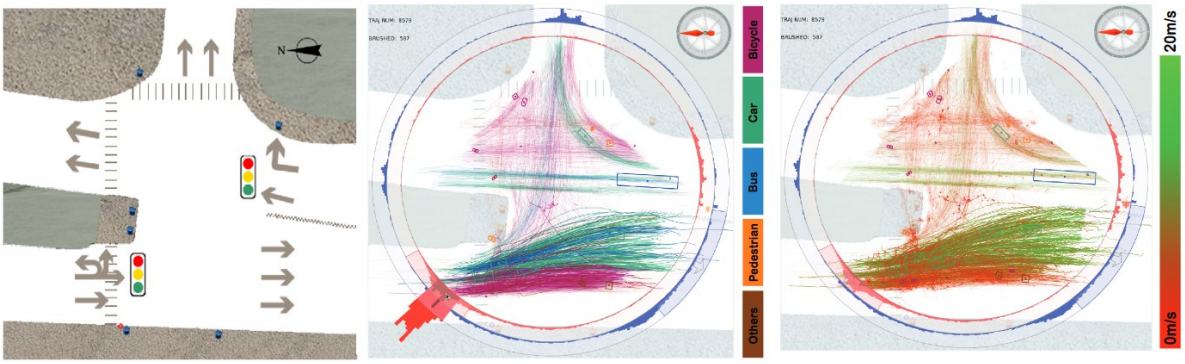


Figure 2.3: TripVista also allows the user to encode several trajectory attributes according to color hue, such as the type of object which the trajectory represents or the speed at which it was moving.

view is also used to show the affluence of different object types over time, overlaid with glyphs representing trajectory direction, and parallel coordinates display several data related to each trajectory (which is also color-coded by object in this view), such as start time, trajectory duration, average, minimum and maximum speed, trajectory distance and angle changes.

Keeping with real-time visualization on somewhat smaller scales, Steptoe et al. (2018) created a visualization to track theme park attendants. This visualization also aggregates attendants into groups (since they will naturally attend the same rides at the same time), using agglomerative hierarchical clustering with a Levenshtein distance function.

Moreover, since this is a theme park and attendant trajectories will show them remaining at the same place for some time, spatial events are also analyzed (that is, attending rides). Due to this, preprocessing not only considers trajectories as trajectories but partly as a series of spatial events.

This visualization gives a lot of control to the user: they can freely select any attendant ID and customize the tolerance for group detection (Figure 2.4 (1)). Trajectories belonging to the selected IDs will be displayed on the map in Figure 2.4 (2), and pressing play will animate them, with a time slider for brushing over or selecting a specific time. When showing users, the software has considered a group, the most representative trajectory will be shown.

A calendar heatmap view (Figure 2.4 (3)) displays attendance in each ride for each hour of one day.

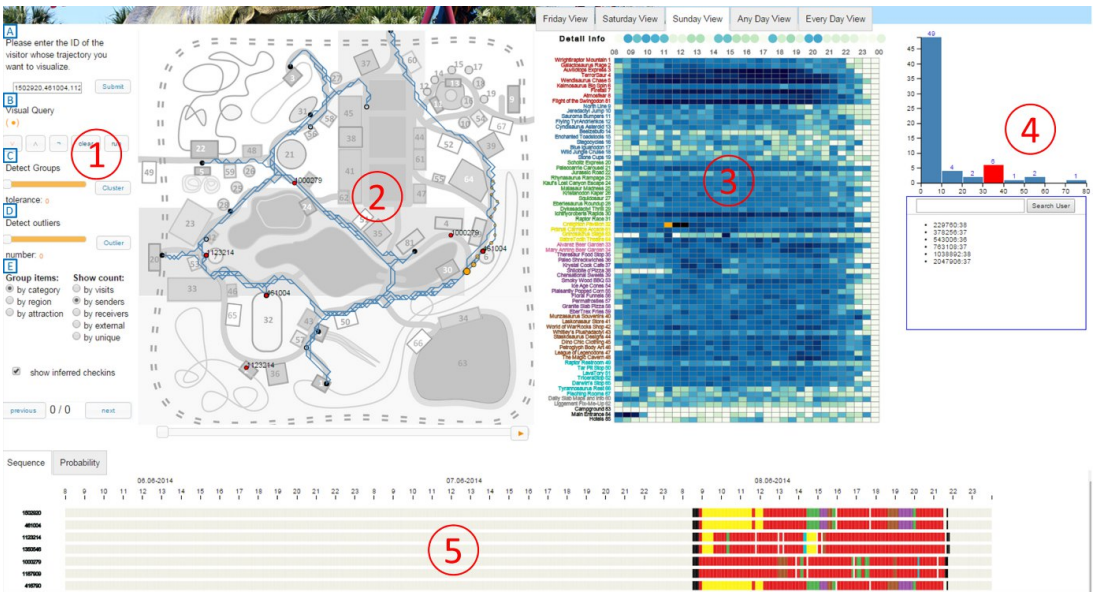


Figure 2.4: The interface in the theme park dynamics visualization from Steptoe et al. (2018).

A bar chart (Figure 2.4 (4)) displays how many attendants (in the y axis) are making certain numbers of communications (in the x axis), and an event sequence view (Figure 2.4 (5)) shows which rides were being attended by which ID at different times.

The lack of scientific research in real-time geospatial visual analytics begs the question of what is used in entities such as transport companies. Cruz and Gonçalves (2014) discuss several case studies pertaining to systems currently used, one of those being the monitoring system at Carris, Lisbon's biggest public transport company.

Carris uses two monitoring systems: SAEIP and XTRAN. SAEIP is focused on viewing bus lines in a non-geospatial manner (which is not very relevant for this work), but XTRAN complements it with a map view. XTRAN is, therefore, more relevant, but it is somewhat limited compared to the research that we discuss in this section. For example, while XTRAN can display any vehicle's position history (up to 6 months), it only shows isolated points with little detail.

Clustering builds groups (clusters) from objects that have similar properties (Andrienko et al., 2013). In spatial clustering, this would mean spatially close objects and/or objects with similar properties, such as shapes or spatial relations among components.

The aforementioned works do not extensively discuss the techniques used to run these real-time visualizations. OCluST (Mao et al., 2018) is an incremental algorithm for the online clustering of trajectories, containing a micro-clustering component intended to cluster and summarize the most recent sets of trajectory segments and a macro-clustering component that builds large macro-clusters out of micro-clusters.

We can find additional works if we look beyond geospatial data. VisBig (Fialho and Gonçalves, 2018) is a data visualization based on the concept of graceful degradation. This concept consists of progressively aggregating and simplifying data as it grows older, allowing a user to focus on the most recent data while still having the older data's context and being able to compare newer and older data.

In this work, data is divided into different modules, each module representing data in different ways. These modules are represented side by side, with data flowing from right to left as new data is streamed in, transitioning between the different modules. Each module shows data in a more simplified way, from a scatterchart which shows the more recent data (Figure 2.5, right), to a linechart that begins to simplify it (Figure 2.5, middle), to a bar chart that simplifies it even more (Figure 2.5, right).

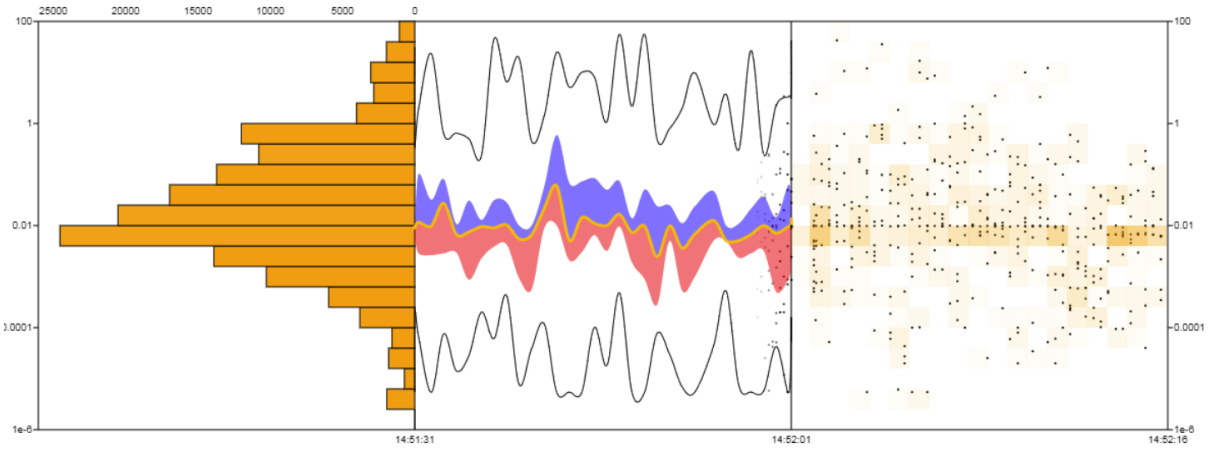


Figure 2.5: Visualization in Fialho and Gonçalves (2018). Data is gracefully degraded from right to left.

2.2.2 Large Amounts of Data

Should we remove the constraint of real-time in visualizing large scale geospatial data, we will find many more works. In these works, we find a greater variety of methods being employed. Multiple coordinate views (combining standard maps with additional multivariate or temporal views (He et al., 2019)) are often used, as they can help the user further their understanding of the problem at hand with non-spatial information being encoded in simpler ways to read.

Something we will discuss throughout this section is aggregation. Aggregation can be useful for dealing with large amounts of data, distilling noise, and enabling an overall view of multiple movements' spatiotemporal distribution. Clustering algorithms function well in loading large amounts of data, which facilitates displaying macro information without data loss (Andrienko et al., 2013).

One such example is Vaite (Yang et al., 2019), which focuses on exploring where trajectories start and stop. Vaite is built upon Spark [1], which is inherently designed for big data analytics applications. Such start and stop events are aggregated into bubbles for each region, the result being a bubble chart with bubble positions overlaid on the map, as seen in Figure 2.6 (a). These bubbles depend upon the user-defined zoom level; different zoom levels may show cities or city neighborhoods as bubbles, for instance.

If the user is not concerned with concrete trajectories, the star topology view in Figure 2.6 (b-1) will display relationships between each bubble. The user can also select a rectangle and view trajectories

[1] Spark: `https://spark.apache.org/`, last visited December 30th, 2020

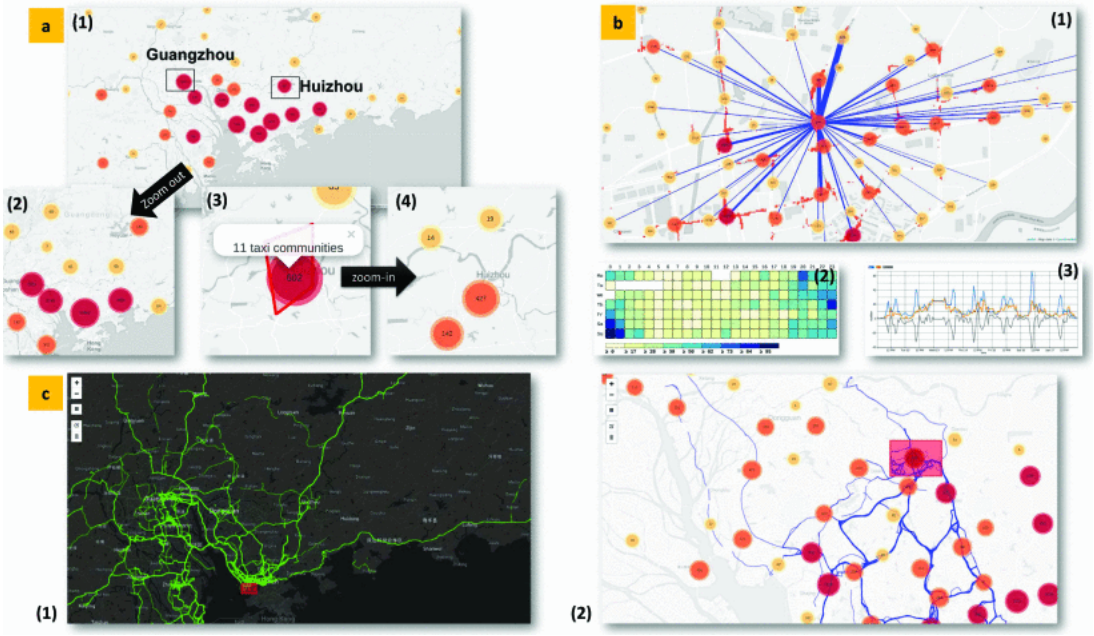


Figure 2.6: An overview of Vaite (Yang et al., 2019), an interactive big urban data trajectory visualization system.

starting from the selected area (Figure 2.6 (c-1)), as well as overlay the bubble chart on top of the selection. (Figure 2.6 (c-2)).

A pixel-based view (Figure 2.6 (b-2)) of traffic jams encodes color to show the number of traffic jams on each time period (x the hour of the day, and y the day of the week). The visualization is not as focused on real-time changes as Gomes et al. (2017), so time series data can be placed in its own separate view (Figure 2.6 (b-3)), allowing the analysis of vehicle flow over time on selected bubbles.

VAUD, a Visual Analyzer for Urban Data (Chen et al., 2018) focuses on letting the user create complex queries with a drag-and-drop interaction system. It allows the user to drag condition nodes and small visualizations to create queries and visualize results on the map view.

The map view itself has a slider for controlling time and overlaid glyphs representing points of interest (Figure 2.7). Point data is aggregated into heatmaps, while trajectories can be represented by polylines or aggregated into heatmaps as well. The query system allows interesting interactions, such as filtering certain vehicles and only showing their trajectories, or filtering trajectories by speed and time of day, letting the user focus on the most congested streets in a city by checking the resulting heatmap.

VAUD can also display space-time cubes (Figure 2.8). These are a good way to display temporal

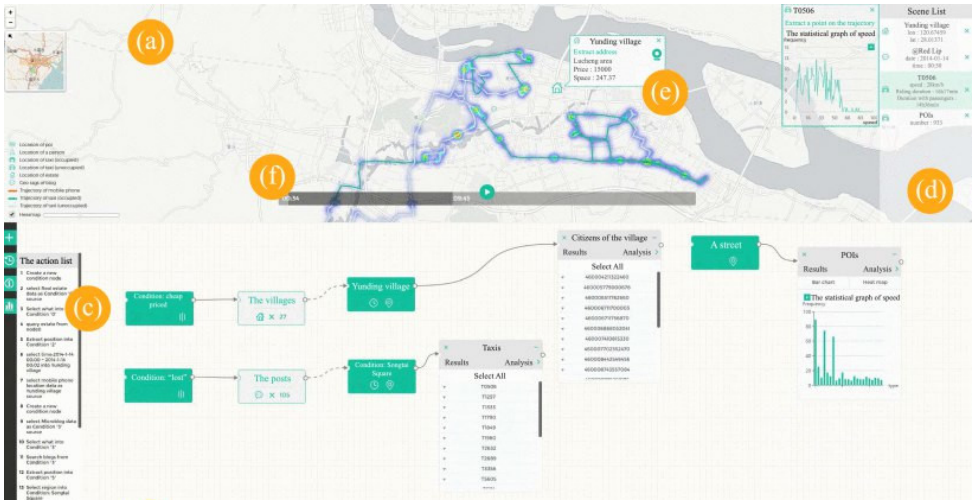


Figure 2.7: Query view and map view in VAUD (Chen et al., 2018).

data, turning a 2D spatial view into a 3D spatiotemporal one by having the z axis represent time. As with all 3D visualizations, however, the space-time cube has problems such as occlusion (lines may be on top of one another, restricting the user from properly visualizing the data (Munzner, 2014)).

Preprocessing in VAUD includes ping-pong effect removal, noise removal, duplication removal, and missing value competition to increase data accuracy since mobile phone location data in this paper's dataset has an average accuracy of 0.269km.

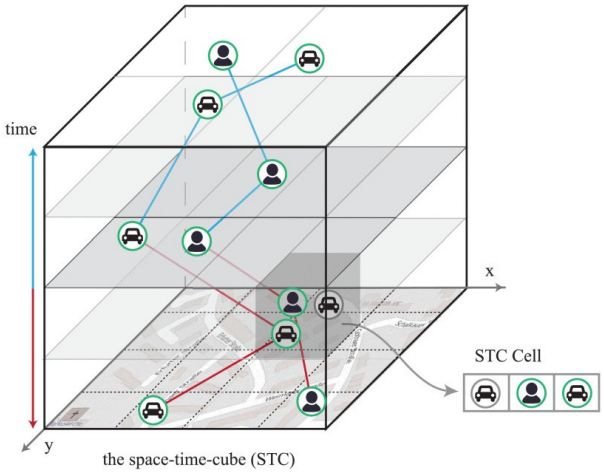


Figure 2.8: Space-time cube in Chen et al. (2018).

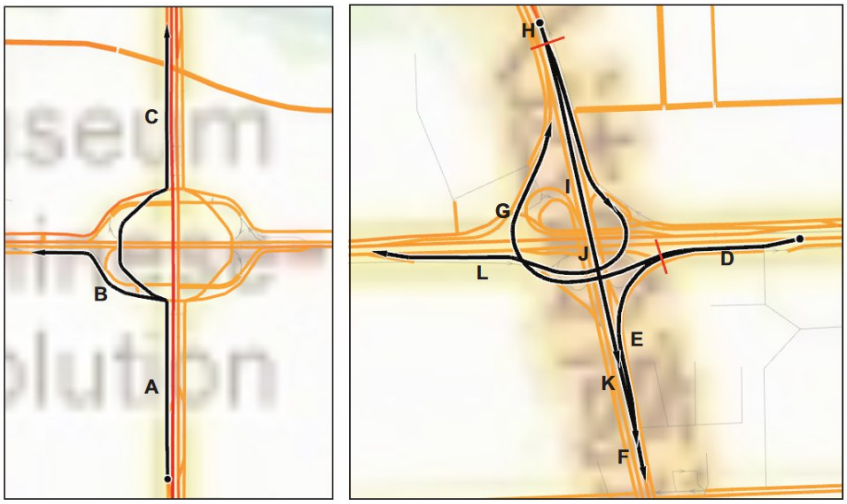


Figure 2.9: Map view in Wang et al. (2013).

Wang et al. (2013) present a visual traffic jam analysis visualization. While the focus is on showing traffic jam patterns, this paper presents interesting views on trajectory visualization.

Preprocessing starts by filtering irrelevant data, merging and splitting ways, and correcting errors, partly because road network data from OpenStreetMap includes non-drivable roads such as waterways. The trajectories are cleaned afterward, filtering stops that do not coincide with a traffic jam (such as parking), and removing GPS errors, such as unrealistic coordinates outside certain ranges (this work was specifically about Beijing, China), duplicated time stamps, and unrealistically high speeds (above 90km/h).

Following that, map matching (matching GPS trajectories to the road network) to correlate trajectory and road speed is performed using an adapted ST-Matching algorithm (Lou et al., 2009), and so the result is that each sampled trajectory point is mapped to a position in a directional way (what this work refers to a road and a direction). This is not all, as the speed of ways is computed, and traffic jam events are detected.

The map view in this visualization is distilled to showing road networks (Figure 2.9), with their color hue corresponding to how congested they are. However, traffic jam propagation is also shown in black.

Another interesting way to display road networks and analyze traffic is seen in TrajGraph (Huang et al., 2016), where a graph is generated from massive amounts of taxi trajectory data, representing an entire city's road network and its traffic information.

In these graphs, streets are vertices, and the connections between them serve as the edges. Differ-

ent vertex weights for the graphs can be defined, such as length, number of taxis, average travel time, or average speed. Graph partitioning is used to create region-level graphs for the user to analyze instead of street-level if they so desire, and centralities, PageRank, and betweenness are computed from traffic information to reveal the roles of certain streets (such as hubs and backbones).

The TrajGraph visualization consists of a graph view (Figure 2.10 (2)), a time series view (Figure 2.10 (3)) and a map view (Figure 2.10 (4)). The graph view shows the region level graph, allowing the user to click on a node and select the corresponding region in the map and time series views. The time series view shows a line chart that displays temporal changes in centrality scores and general traffic information (such as average speed, average travel time and traffic flow). The map view allows the user to study traffic information at a street level, and a radial chart can be turned on, showing how this information evolved over time.

According to Rinzivillo et al. (2008), a density-based clustering algorithm such as OPTICS (Ankerst et al., 1999) can be implemented in a way that separates the process of finding clusters from the distance between trajectories. One of these distance functions is called route similarity. This algorithm repeatedly searches for the next pair of closest positions and computes the mean distance between positions, adding a penalty for unmatched positions (that have been skipped as insufficiently close between each other).

OPTICS considers two objects as neighbors if the distance between them (computed by route sim-

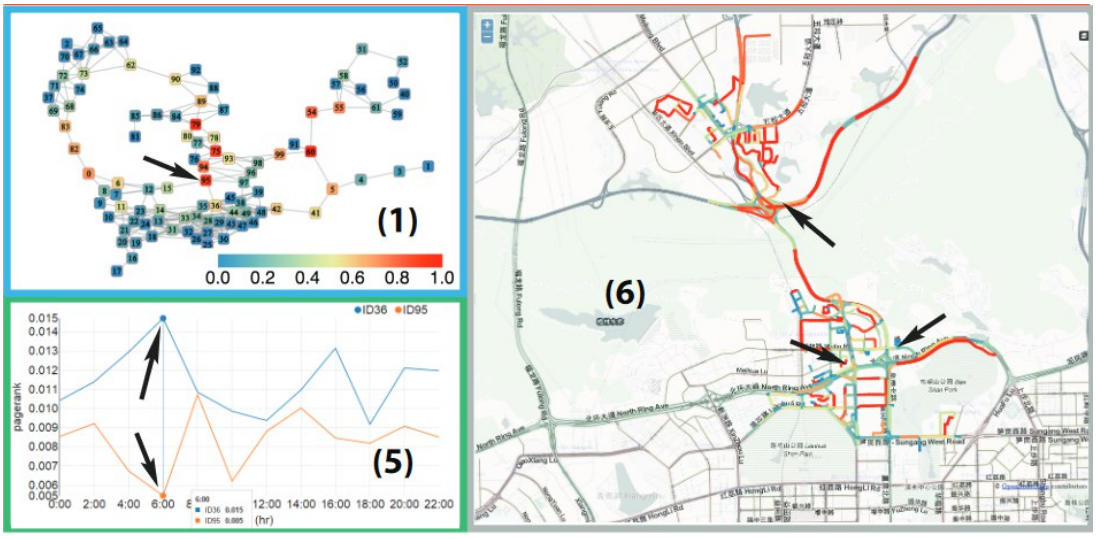


Figure 2.10: Interface in TrajGraph (Huang et al., 2016).

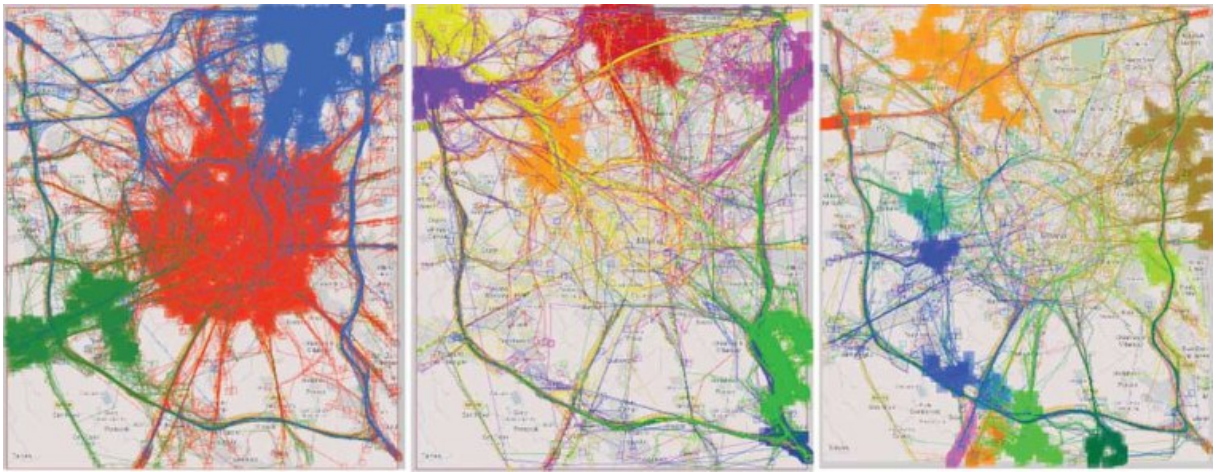


Figure 2.11: Clustering trajectories of Milan traffic. On the left, the three biggest clusters. In the middle, clustered trajectories after excluding the three biggest and changing parameters, and on the right, clusters after excluding the previous clusters as well.

ilarity) is below a threshold. The algorithm tries to find an object with a minimum number of neighbors and takes it as a seed for a new cluster, adding to this cluster all the object's neighbors, neighbors of neighbors, and so on, until there are no more neighbors of cluster members. Afterward, the algorithm finds a new seed and repeats the process until no new seeds can be found. Figure 2.11 contains some examples of clustering trajectories using OPTICS and route similarity with different parameters.

More recently, Patwary et al. (2013) improved upon OPTICS with the scalable and parallel POPTICS. Afterward, Deng et al. (2015) extended POPTICS to scalably cluster trajectory data and implemented this extended algorithm on Kepler GPUs, using the Kepler architecture's Hyper-Q feature.

For a work that focuses more closely on the trajectories themselves, and in different aggregation methods, we have TrajAnalytics (Zhao et al., 2016). TrajAnalytics is open-source software with a BSD license, and for developing this software, Zhao et al. used two datasets: an OD dataset from taxi trips in New York and complete taxi trajectory data from Porto, Portugal.

This visualization uses `d3.js` and `leaflet.js`, and allows viewing raw trajectories (Figure 2.12, left), heatmaps (Figure 2.12, center) or welded graphs (Figure 2.12, right), allowing the user to explore different aggregation options. The user can make simple polygon, rectangle or circle selections and decide whether the selection should display trajectories starting, ending or intersecting with the selection.

Having already discussed heatmaps and welded graphs, we can move on to another form of trajectory data aggregation. Density maps are created by visualizing density fields made by aggregating

Figure 2.12: Selecting trajectories starting in Praça Mouzinho de Albequerque, Porto (roundabout in the center). Start points in green and end points in red. From left to right: raw trajectory data, heatmap and welded graph. From TrajAnalytics (Zhao et al., 2016).

smoothed trajectories. In Scheepens et al. (2012), these are rendered with color mapping and illuminated height fields; color saturation is used to encode the value of a field, while different hues represent different fields.

Hardware acceleration is used to compute these density fields. The implementation starts with constructing a texture to represent a grid in geographic space and then rendering a polyline with geographic coordinates to this texture for each trajectory. A geometry shader is used to construct a bounding box

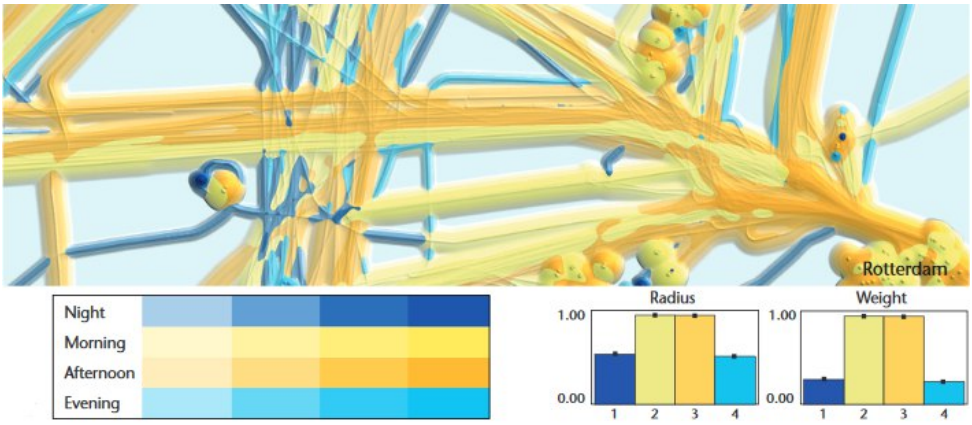


Figure 2.13: An example of user interaction in Scheepens et al. (2012): selecting different color hue and density field radius and weight for different times of day.

Figure 2.14: The spatiotemporal view in Chen et al. (2016).

that fits the density kernel radius, and a fragment shader to evaluate the density function for each line segment.

The use of shaders means graphics hardware computes density in parallel, rendering density maps in a split second and enabling real-time interaction. The user can interactively change parameters such as color, radius, or weight of different fields, such as the example in Figure 2.13, where different parameters are used for different times of day.

As we have seen throughout this section, a lot of non-spatial information is displayed in different views, which has the downside of requiring the user to focus on multiple views and reducing the map view's screen space. However, viewing temporal information such as time series can be integrated more closely into the main map view.

Chen et al. (2016) describe a visualization of movement patterns from geo-tagged social media data. While the visualization does not focus as much on trajectory data (it is more related to estimating movement and trajectories from simpler geospatial data, and does not display raw trajectories so much as it focuses on the number of movements between points of interest), it integrates time series into the map itself (Figure 2.14), allowing us to discuss a more interesting kind of filtering.

Geo-tagged posts are aggregated, and movement between places is estimated and also shown. Around the place of interest and movement display, however, we can see a radial chart where the user can visualize the distribution of posts between different places, as well as the number of movements both in and out of them (the user can specify if the chart displays 24 hours of one day or seven days of

Figure 2.15: The spatiotemporal view in Sil and Gonçalves (2018). On the left, flat color representing the average of time in each line. On the right, gradients represent the distribution of times in each line. In both figures, width encodes the number of lines in each bundle line.

a week).

Edge bundling is a visualization technique that clusters similar lines and represents them all as only one line, which makes a visualization look much cleaner while showing the same data. Usually, this technique is used in chord charts and maps that display OD data, that is, sets of lines otherwise disconnected from each other. However, Understand My Steps (Sil and Gonçalves, 2018) uses edge bundling with trajectories, sets of lines that do have connections between each other (we will distinguish this type of edge bundling by calling it ”trajectory bundling” henceforth).

Understand My Steps presents four different ways to visualize data, three of them using trajectory bundling. As in most edge bundling visualizations, width is used to represent the number of lines contained in each bundled line. However, Understand My Steps also uses color to represent time (specifically, how old a bundled line is). This time representation can take the form of a single color representing the average age of the bundle (Figure 2.15, left), or a gradient that represents the distribution of time over a bundle (Figure 2.15, right).

2.3 Discussion

Visualizing large amounts of trajectory data, especially when it comes to doing so in real-time and with a streamed data source, is a fairly recent field of study. The table below summarizes what each work features, and we will discuss said features in more detail afterward.

Real-time visualization is not featured frequently in the discussed works, and if we look at the works that do feature it, we can see why. Gomes et al. (2017) may be real-time, but it does not display complete trajectories. TripVista (Guo et al., 2011) and Steptoe et al. (2018) display complete trajectories, the tradeoff being that they focus on much smaller scales (Guo et al. (2011) focusing on a city intersection, and Steptoe et al. (2018) on a theme park). Fialho and Gonçalves (2018) is a system that handles both real-time streaming data and large amounts of it, but the data is spatial.

Streaming data sources are only mentioned in Gomes et al. (2017), where the proposed solution is performant enough to work with streamed data due to the use of hot motion path discovery of Sacharidis et al. (2008).

Scalability is an important issue and something that is much more discussed than visualizing data in real-time or streamed sources. Many authors have the goal to visualize big geospatial data, and so almost all the discussed works are scalable and handle large amounts of data except for Guo et al. (2011), and Steptoe et al. (2018), which is the tradeoff for these works featuring real-time visualization, as we discussed above.

Different authors use different methods to display these large amounts of data and maintain readability: Gomes et al. (2017) only shows the points and a small trail, Chen et al. (2018) heatmaps the

Work	Real-time	Streaming	Scalable	Time	Spatial
Gomes et al. (2017)	×	×	×	-	×
Guo et al. (2011)	×	-	-	×	×
Steptoe et al. (2018)	×	-	-	×	×
Yang et al. (2019)	-	-	×	×	×
Chen et al. (2018)	-	-	×	×	×
Wang et al. (2013)	-	-	×	×	×
Huang et al. (2016)	-	-	×	×	×
Zhao et al. (2016)	-	-	×	×	×
Scheepens et al. (2012)	-	-	×	-	×
Chen et al. (2016)	-	-	×	×	×
Rinzivillo et al. (2008)	-	-	×	-	×
Sil and Gonçalves (2018)	-	-	×	×	×
Fialho and Gonçalves (2018)	×	×	×	×	-

Table 2.1: A summary of features in our related work.

trajectories, Wang et al. (2013) and Huang et al. (2016) use the raw trajectory data to build a road network visualization. Scheepens et al. (2012) builds density fields with the raw data to then display, Chen et al. (2016) only shows the number of people moving between points of interest, and Rinzivillo et al. (2008) uses clustering techniques. Yang et al. (2019) shows the trajectories, but the user can also simplify them with the star topology view we discussed, and Zhao et al. (2016) allows the user to select from raw trajectory data, heatmap, or welded graph. Sil and Gonçalves (2018) uses edge bundling to represent multiple lines as one.

Finally, visualizing **time** is another important feature when visualizing trajectory data, and most works do display time, though each in their own ways. Gomes et al. (2017) focuses on displaying points and trails, Scheepens et al. (2012) on displaying the density fields, and Rinzivillo et al. (2008) on clustering trajectories, so they do not visualize time series. Guo et al. (2011) uses ThemeRiver to visualize types of vehicles over time, Steptoe et al. (2018), Wang et al. (2013) and Yang et al. (2019) use pixel-based views, Chen et al. (2016) and Huang et al. (2016) use radial charts, Sil and Gonçalves (2018) use color, and Chen et al. (2018) and Huang et al. (2016) use line charts to visualize time.

As we have seen throughout this chapter, there is a great variety of papers that address subjects that interest us: how to represent space and time and doing so in real-time, with streamed data, and in a scalable manner. However, the state of the art lacks something that manages to do all of these things. Some papers present real-time solutions but do not deal with large amounts of data, while others will deal with it but not handle visualization in real-time as the data is streamed in.

Chapter 3

FastGeo

In this section, we present the different visualization techniques and the data processing algorithms that support them. These are used in order to study the FastGeo concept, which we will start by explaining.

As we have already discussed, this work aims to visualize large amounts of data in real-time. The FastGeo concept consists of dividing time into different periods and representing each period using different visualization techniques. As data passes from one period to another, it will be gracefully degraded (Fialho and Gonçalves, 2018), aggregating trajectories and gradually reducing the amount of memory being used.

3.1 Time Periods

If we are to visualize large amounts of data as they are streamed in real-time, we will quickly realize that we cannot keep the entire detail of every single datum we receive. Not only will this result in a visualization that is too dense and full of unprocessed data, but we will consume far too much memory storing this unprocessed data. Therefore, we must find a way to gradually simplify the data to save on memory and keep a readable visualization. This is where the concept of time periods comes into play.

Time periods allow us to group the data that is received based on different factors, displaying each group differently to differentiate these periods easily. As data passes through each period, it loses detail, allowing it to occupy less memory without losing so much information that it becomes impossible to understand (Figure 3.1). The idea is to gracefully degrade data as it becomes older and, therefore, less relevant individually (older data is still relevant, of course; however, we are more interested in the

Figure 3.1: Graceful degradation of trajectory data. On the left, raw trajectories. In the middle, bundled trajectories. On the right, grid binned trajectories. The data is gradually simplified as it gets older (left to right).

overall trend, as opposed to each datum).

The time period concept serves to simplify data gradually, following the graceful degradation concept. Naturally, we want to do this as the data gets older since this is a streaming visualization, and we are more concerned with exactly what is happening currently than the exact happenings of something that happened two years ago. The most recent data, therefore, is what should have the most detail possible.

After consideration, we conceived three periods: an ongoing period, a slightly more simplified recent period, and, finally, a further simplified history period.

The **ongoing period** is the most detailed, containing trajectories as they are received. It corresponds to trajectories that have not ended: if an object stops, its trajectory until that point will no longer be part of this period. If it starts to move again, then its trajectory, starting with the point where it last stopped, will be part of the ongoing period. We will discuss how a stop is defined and detected in Section 4.2.2.

After trajectories stop, they will enter the **recent period**. This period covers an adjustable period of time (e.g., stopped trajectories to one hour ago, or stopped trajectories to one day ago). Instead of keeping a trajectory's complete data, we will use trajectory bundling (joining similar lines together) to reduce memory usage and visual clutter (which we will further discuss in Section 3.2).

We can still simplify the data further by not keeping separate lines in memory. The **history period** is the final accumulator for the data, and every line that is older than the recent period's limit is represented here. Rather than storing lines like in other periods, in the history period, we fit each line into the squares of a grid encompassing the entire dataset's boundaries. We will discuss this grid binning process in

Section 3.3.

3.2 Trajectory Bundling

Trajectory bundling is a process where a set of GPS tracks are simplified down to a set of lines (essentially, pairs of points). This process makes it so that any overlapping lines are joined, and the resulting lines contain data pertaining to how many objects they represent. This data, the number of objects a segment represents, will be referred to as **total** below. The visual result is something similar to some of the works we have discussed (Huang et al. (2016), and Wang et al. (2013)), where each street only has a color representing how congested it is, with minimal line overlap.

As we have already seen in Chapter 2, Understand My Steps (Sil and Gonçalves, 2018) uses trajectory bundling in a static visualization and dataset, and its methods are well suited for that work's constraints. Since it will perform trajectory bundling only once, it can afford to compare every track segment to every other track segment, resulting in a very accurate (though not very fast, which does not matter in its case, as we have already stated) result.

However, we are more constrained. Since new data is added every few seconds, trajectory bundling must be performed at the same rate. Moreover, this data may constantly be accumulating, meaning that even if the algorithm is extremely optimized, it will eventually become too slow as more data is streamed in, making it so that bundling cannot be performed at a steady rate.

Therefore, there were two important problems to address when creating our trajectory bundling algorithm: we want to focus on efficiency, to make sure we can process data as fast as possible, and data filtration, to have it deal with the least data possible and avoid the necessary time for its execution from growing too much as the number of data increases. Of course, we want to tackle these two problems, but not at the cost of the results' quality.

This algorithm deals with two sets of data. Below, we will refer to lines that are being added to the corresponding time period as **new** data and lines that already were in the time period as **old** data.

3.2.1 The Main Algorithm

We started by using the algorithm detailed in Understand My Steps (Sil and Gonçalves, 2018), but it was not fast enough for the constraints. We reworked both the comparison and how lines are merged and added a data filtering technique that reduces the number of comparisons made to get a satisfying performance. We will discuss this performance and compare it to performance with the Understand My

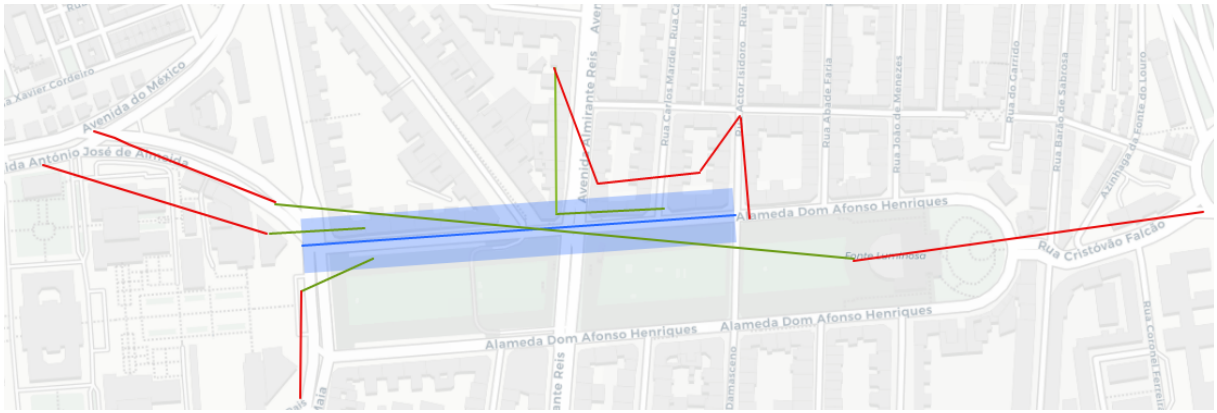


Figure 3.2: A new line (blue) selecting old segments near it (blue rectangle). Green segments are selected and compared with the new line, whereas red segments are ignored.

Steps algorithm in Section 5.1.

3.2.1.1 Data Filtration

The first step in this algorithm is to minimize the number of comparisons performed, as these comparisons are computationally expensive to perform. Naturally, only segments that are near to each other and with similar bearing (that is, the angle between the segment and north) can be joined, as segments that are too distant will consist of overly different trajectories (different streets, for example), and segments whose angles are too different will likely consist of different trajectories as well (for example, segments on intersections or roundabouts may be close to each other, but their bearings are likely different).

Therefore, we want each new segment to select old segments that are close to it (similar to Figure 3.2). Each new segment will then only compare itself to the segments it selected. This avoids comparisons with the rest of the old data, and greatly reduces the number of comparisons made.

Before beginning comparisons between two lines, we perform some additional checks. As we have stated, only lines with similar angles are worth comparing. Therefore lines that do not comply with this condition are not compared (the exact angle differences are user-configurable, as is whether to let opposite lines be joined). Additionally, when we compare new data with itself, segments that come from the same object are not compared.

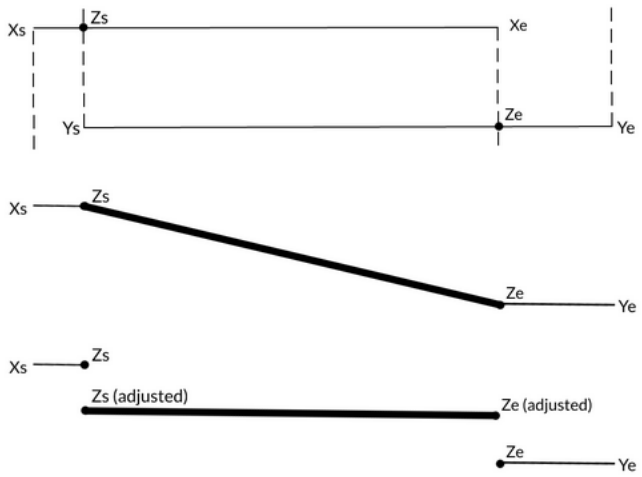


Figure 3.3: An example for the comparison algorithm. At the top, using normals to find intersections. In the middle, the resulting merged line without adjustment. At the bottom, the adjusted line.

3.2.1.2 Comparison

The most computationally expensive task in the algorithm is comparing two segments and checking if they are eligible for merging. To do this, we try to intersect the normal line (n) of one segment (x, points x_{start} and x_{end}) with the other segment (y, points y_{start} and y_{end}). We test both start and end points between x and y. If a point cannot be found, then we test between y and x.

The resulting intersection points (z_{start} and z_{end}) are then tested: to be accepted, they must be inside y (as when we test an intersection point, we are using infinite lines and not segments - simply checking whether the intersection point is inside a bounding box of y suffices), and the distance between them and the point in x we used to compute the normal must be below a threshold.

If both conditions apply, additional adjusted points (Figure 3.3, bottom) are created by computing a weighted average (the weights being the number of segments being bundled in x and y) between the intersection point and the matching point in x.

These adjusted points are used to prevent overly angled lines in cases where the start point is from one segment and the end point from another (Figure 3.3, middle). These angled lines may cause problems, as they then cannot be bundled further due to being too angled to merge with other lines (see Figure 3.4).

3.2.1.3 Merging Lines

After we make comparisons and acquire start and end interception points (should they exist), we use these to determine if and how to merge these lines. Only if both start and end points were found do we bundle anything. In that case, we have to check if any of them is an extremity (the comparison method we described above returns this information).

We only create segments other than the unification of x and y if one or both points are not extremities, as if both points are indeed extremities, the lines are either parallel with parallel extremes or the same line, and no partial segments for any of these lines require creation. If one of the points is not an extremity, then a segment is created to represent the part of a line that was not merged (see Figure 3.5).

Optionally, the partial lines may also be adjusted. This is an attempt to keep lines from becoming too distant. It improves the overall visual quality, and keeping segments close together helps avoiding issues where some lines become too distant to bundle. The adjustment is done by computing an intersection point between the normals of the non-bundled segments and the bundle (Figure 3.6).

3.2.1.4 Double-Pass Method

We perform the above algorithm twice.

The first pass performs it between new data and new data. This is done to ensure that there is no segment overlap in this set of data before the second pass. There is no data filtering based on the distance between segments in this pass, as data filtering is only done between new and old data, and new data is minimal compared to old data, therefore not requiring such strict filtration.

The second pass compares the already bundled new data with old data. Since the new data was

Figure 3.4: Unadjusted (left, red) versus adjusted (right, green) line. The unadjusted line may differ too much from other segments (in black) when it comes to angle.

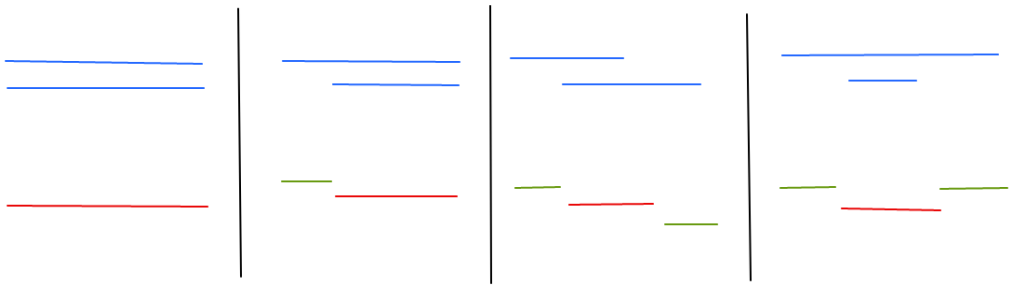


Figure 3.5: If part of a segment does not perfectly match the merge, a new partial segment is created. Original segments in blue, partial segments in green, merged segments in red.

previously bundled, then it may only overlap old segments. Old segments have already been bundled previously. Therefore, the only lines that may overlap are pairs of old and new, ensuring that there are no overlapping segments after this pass.

3.2.2 Adjacency Adjustment

While the trajectory bundling algorithm described so far is efficient, it has one problem. It does not take adjacencies into account, meaning segments are not aware of whether to remain connected with each other. The visual result is lines that are scattered throughout the map with no connection, making paths harder to follow. Therefore, taking adjacencies into account is another important part of our trajectory bundling solution.

When lines come from the ongoing period, they are simplified using the Ramer-Douglas-Peucker

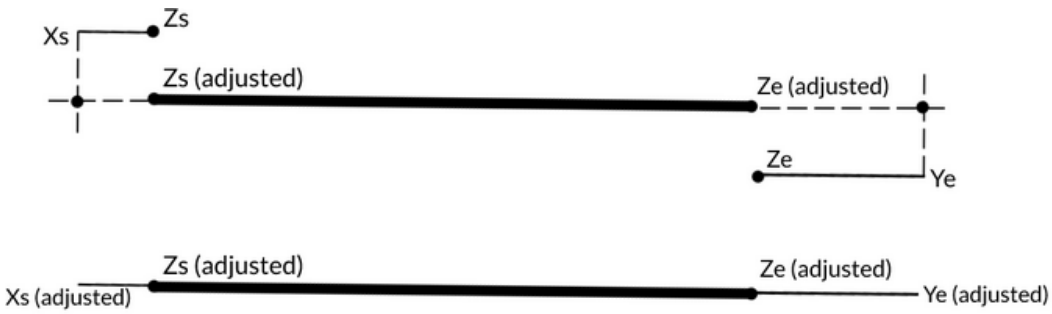


Figure 3.6: Additional adjustment for partial segments.

algorithm, which reduces the number of points in a line (Saalfeld, 1999). The resulting line is a list of points, each pair representing a line segment. We split all of the given lines into segments to form the new segment set that will be used for trajectory bundling. When we do this, we give an ID to each segment, and in each point, we store the segment ID it is adjacent to (if it is adjacent to any line). Figure 3.7 provides an example.

The old segments also have this adjacency data. When segments merge, a new segment using the merge is created. This segment will have a new ID, but it inherits all adjacencies. This ensures that the original adjacency data will not be lost no matter how many merges occur. If a partial merge is done, several segments will be created, but they will keep the adjacency data correct according to how this partial merge is done.

At the end of the trajectory bundling process, we need to correct the adjacency data: many older segments will have adjacencies meant to match segments that no longer exist, as they have been merged. We check the new segments, access the segments they are adjacent to, and correct their data to match the merged segments.

The adjacency adjustment itself is a separate part of the trajectory bundling algorithm, as it is only called depending on the period's display method, and it does not replace the non-corrected data. This is done because, as we will see in Figure 3.8, this adjustment often changes segment angles drastically, and since we are using these angles to filter data, it would result in visual problems (segments overlapping because they do not merge properly due to having too different an angle).

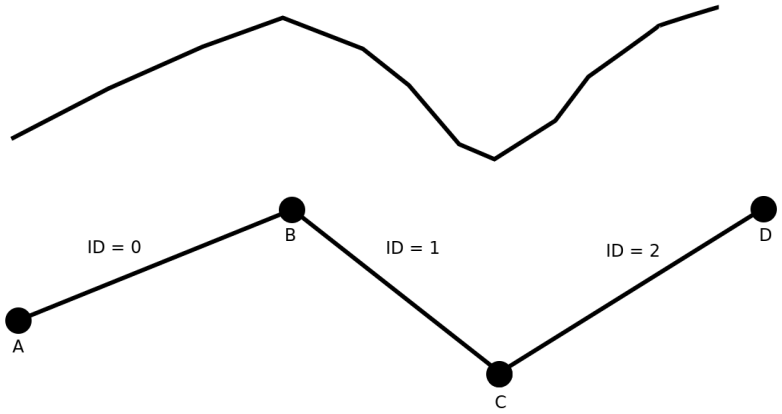


Figure 3.7: On top, track before simplification. On the bottom, track simplified with IDs. Points A and D do not have any adjacency data. Point B in segment 0 is adjacent to point B in segment 1 (and vice-versa), and point C in segment 1 is adjacent to point C in segment 2 (and, again, vice-versa).

To adjust segment adjacencies, we go through all segments that are meant to be displayed. When a segment point has adjacencies, we make a weighted average point using the point itself and all points it is adjacent to (the weight being the total in each point's corresponding segment), and we adjust all of the points to match the resulting average point. A point can only be adjusted once, even if its adjacencies were not directly checked.

3.2.3 Temporal Binning

To simplify the data in the recent period, we use temporal binning as well. The recent period has an associated bin length. While each segment contains timestamps associated with the segments it contains, these timestamps are further simplified with temporal binning. Instead of having separate timestamps, these are grouped into bins of a given length (e.g., five minutes). The result is that each bin is only composed of a timestamp and its corresponding sub-total. The history period removes temporal data.

Aside from temporal binning, we also experimented with an alternative to the history period. Rather than having one final accumulator (the History period using grid binning, as we will discuss in Section 3.3), we explored the idea of modular periods. The idea consisted of letting the user configure how many

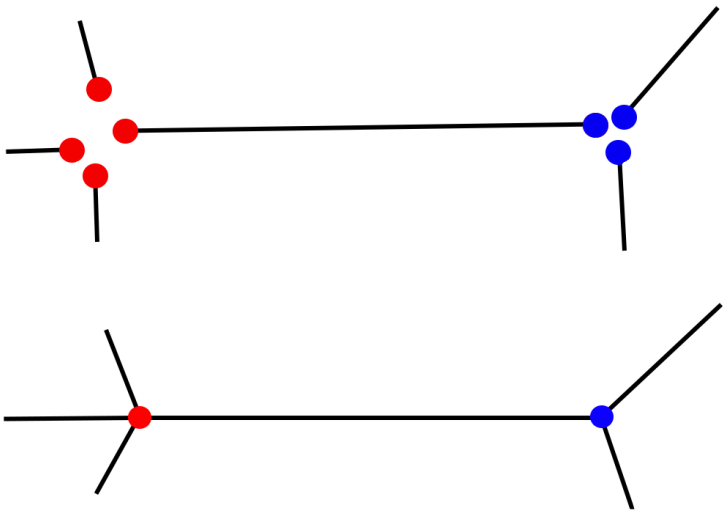


Figure 3.8: Adjacency adjustment (the points in red are adjacent to each other, as are the points in blue). Please note how this adjustment may change the angles of different segments significantly, causing issues if these are to be used in trajectory bundling later.

of these periods they wished to have and then having a simplification mechanism that would gradually reduce the data in each period. We called this simplification algorithm line binning, and the idea was to perform graceful degradation of data in a more controlled and configurable manner.

While temporal binning simplifies temporal data, line binning simplifies spatial data. For each period, a length is defined (the idea being that this length is larger with each period). While trajectory bundling, given an eternal dataset, will eventually split lines (through partial matches) until each line is microscopic, line binning tries to enforce that each line have the defined length, being no longer or no shorter than it.

Due to clarity (having more than three periods shown in the same map will make them hard to distinguish) and performance reasons, we decided to focus on a static set of periods, the last of which using grid binning to simplify the data (something we will discuss in Section 3.3).

3.2.4 Limitations

Naturally, this algorithm has its limitations. While performance limitations will be detailed in Section 5.1, there are other types of limitations to discuss.

First, it is important to note that the quality of the data that this algorithm will produce is heavily dependant upon the quality of the raw data it receives. If the raw data has a very low update frequency, then it will often skip turns. It will send a point update on one street, and then it will send the next point after having already turned, the resulting line between the two points not following the road and instead seemingly going through buildings.

We can only simplify data, not create new points, and figure out where they are supposed to be to return a more correct line. Since these lines are likely to be too angled to be eligible for a merge with the

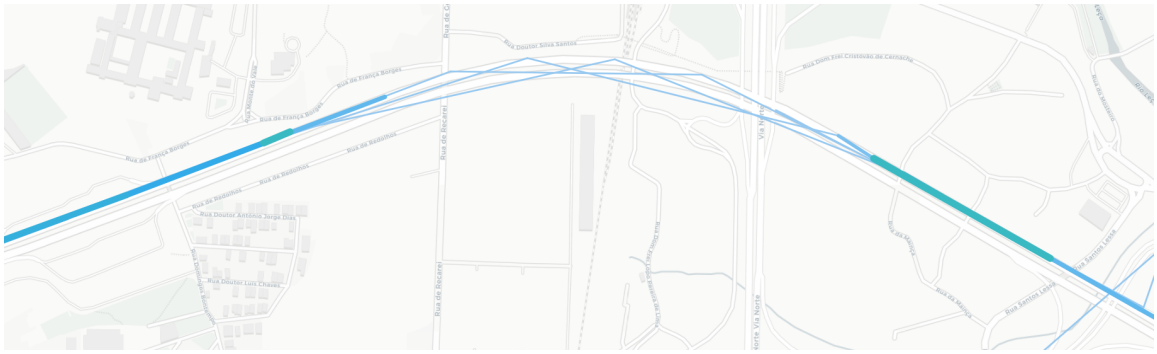


Figure 3.9: Different vehicles will update their positions at different parts of a curve, creating segments that cannot be bundled properly.

rest of each street's lines, they will not be merged and result in clutter when viewed.

While in spaces that consist of mostly straight roads and right-angle intersections (such as grid layout cities, like with most of Manhattan), only a very low update rate will result in problems, spaces with more curves (e.g., highway on-ramps, or roundabouts) are likely always to have unsatisfactory results. This is because, very likely, each object will update its position at different points of a curve.

Of course, this happens in straight roads as well, however in straight roads, the resulting lines are of equal (or at the very least, similar) angles, and therefore this does not affect the bundling results. In curves, however, positions may be updated at different phases of the curve, which results in different enough angles between lines that they cannot be bundled with satisfactory results (Figure 3.9).

3.3 Grid Binning

Grid binning is the most aggressive simplification we use. It serves as a final accumulator for the data because it removes the notion of specific lines from it. Rather than lines, several grids are stored.

We store several grids because we want to show different resolution grids depending on the visualization map's zoom level. Given a configurable resolution (that is, the size of each square), several grids are created where each grid's resolution is a product of the given resolution with a power of two (e.g., a 15m resolution will result in one grid of 15m, another of 30m, and another of 60m). Each square has the number of lines intersecting it (its total, like in trajectory bundling).

To fit a line into the grid, we divide the line into interpolated points, the distance between each point being equal to the grid's resolution (Figure 3.10, middle). For each point, the corresponding grid square is found, and 1 is added to that square's total (Figure 3.10, right). We only fit lines into the most high-

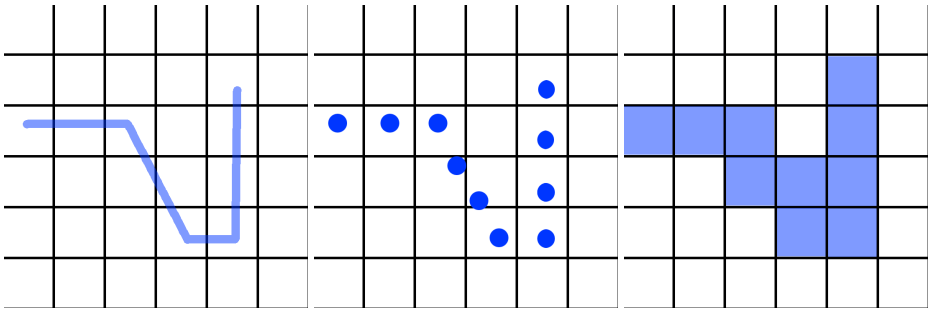


Figure 3.10: Lines being binned into the grid. On the left, the starting lines. In the middle, the resulting points from interpolation. On the right, the resulting grid.

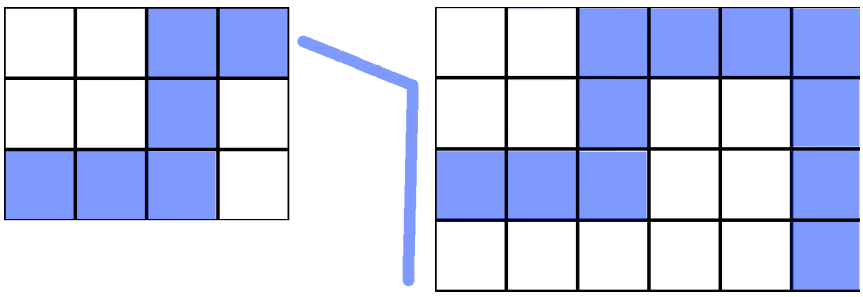


Figure 3.11: Grid expansion. On the left, a new line is outside the grid, which, in turn, expands to incorporate it (right).

resolution grid: the other grids are updated from this one.

The grid is gradually expanded as more data comes into the history period. To expand the grid, a configurable minimum number of lines must exist outside of it: these outliers are stored until this number is reached, in which case the grid is expanded (Figure 3.11). It should be noted that outliers that are too far away from the grid are not considered in this number, as we want to avoid expanding the grid immensely for just one segment.

3.4 Visualization

After discussing the various concepts relating to how data is treated, we will now discuss how it is represented. Below, we will detail the different display methods used in representing the data and other visualization-related specifics.

Depending on the time period, several methods can be used to display data. Much like in data processing, the more data we have, the simpler it should be. Of course, the visual side concerns are not purely performance: we also want to make sure that the display method is not only clearly understandable by itself but works in tandem with other periods and their methods being overlaid on top of each other.

3.4.1 Ongoing Period

Should we follow the aforementioned line of reasoning that periods with more data should be displayed more simply, we can conclude that the period with the least data requires the least simplification.

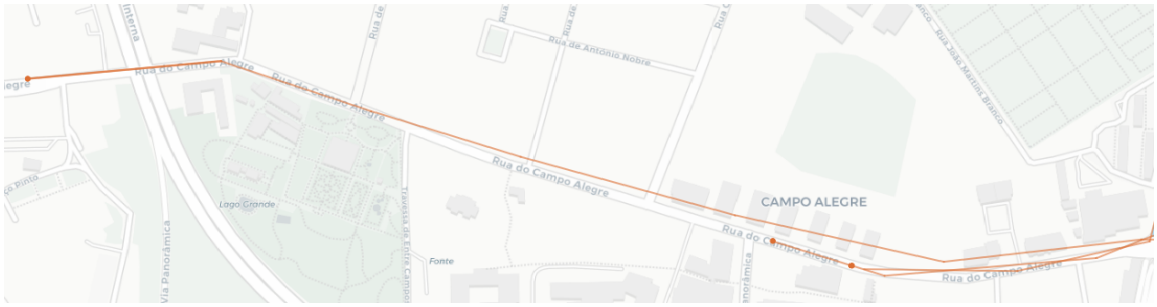


Figure 3.12: The ongoing period's display method.

The ongoing period's lines are shown as they are (Figure 3.12), with no further corrections or simplifications applied.

Since any new data on this period is data that has just happened, we want to make it clear to the user that it corresponds to something that is currently moving. To do so, we animate new lines: rather than appearing in their entirety immediately, the animation mimics a movement from one point of the line to the other.

A small circle is also shown for each separate polyline in the ongoing period. This circle shows each object's last-updated position and implicitly allows the user to know how each polyline is moving since the farthest a segment is from this circle, the older it is.

The size of both lines and circles adapt to how zoomed in the map is. This is done so that lines are never too thin or too wide for whatever zoom level is on the map at any given moment, avoiding occlusion problems (which happen if the map is zoomed out and lines are too wide, covering each other) or lines being too small when zoomed in (since if a user zoomed in they would probably like to look at a line in more detail).

3.4.2 Recent Period

The recent period's data is already simplified with trajectory bundling. This allows us to avoid showing dozens of lines in the same street, which would quickly look cluttered and difficult to understand. Trajectory bundling shows only one line per street, which allows us to explore how to show the amount of traffic on a street in less visually noisy ways.

We use width and color to represent this attribute (Figure 3.13). Each segment, depending on its total, will have a color based on a scale that goes from blue to green (something we will discuss further in Section 4.3.2), and the higher its total, the wider it will be. The brighter green helps draw the user's

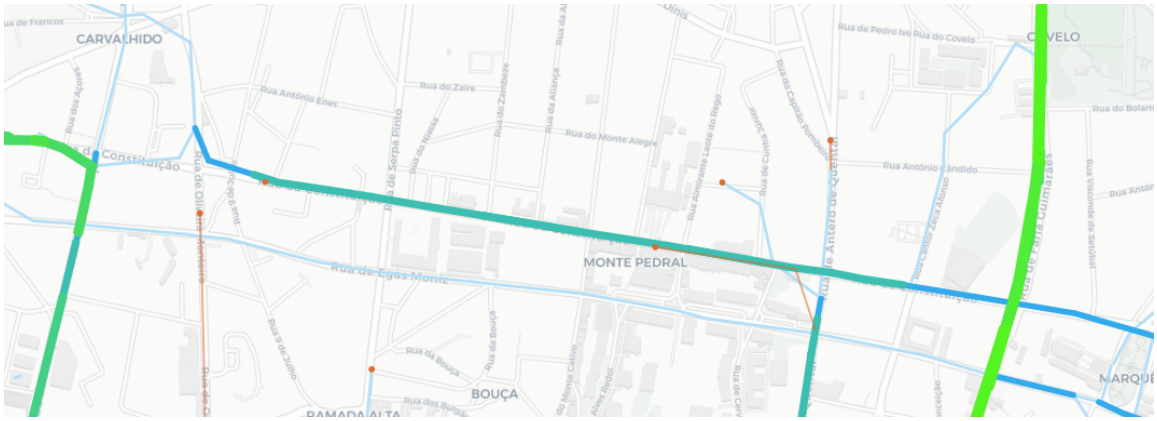


Figure 3.13: The recent period's display method. Greener and wider segments have a higher total.

eye immediately to streets with higher traffic, while width is used to make segments with low total thinner so as not to clutter the screen too much.

These width and color scales are dynamic: as the maximum total varies with time, the scale will constantly adjust to match the maximum to avoid cases where no line reaches the maximum color on the scale, or conversely, where most lines exceed the maximum. Additionally, as with lines on the ongoing period, recent lines adapt to the map's zoom level. The adaptive width tries to keep the widest lines at approximately the width of the streets shown by the map.

3.4.3 History Period

As with the recent period, the history period's data is already simplified, in this period using grid binning. The question here is how to show the square grid bins. To answer this question, we decided to propose two different approaches. Both were tested with users and compared (see Section 5.2).

One such representation uses a cluster heatmap, which consists of a square matrix grid where each grid tile is shaded on a color scale to represent the value of an attribute (Wilkinson and Friendly, 2009). This is a clear fit for grid binning, as it simply consists of shading each grid square in a different color (Figure 3.14, left).

Whereas the cluster heatmap presents data in discrete squares, the spatial heatmap creates continuous surfaces from interpolating discrete points. This is also a good fit for grid binning, as each grid square can be represented as a point and then interpolated into one such continuous surface (Figure 3.14, right).

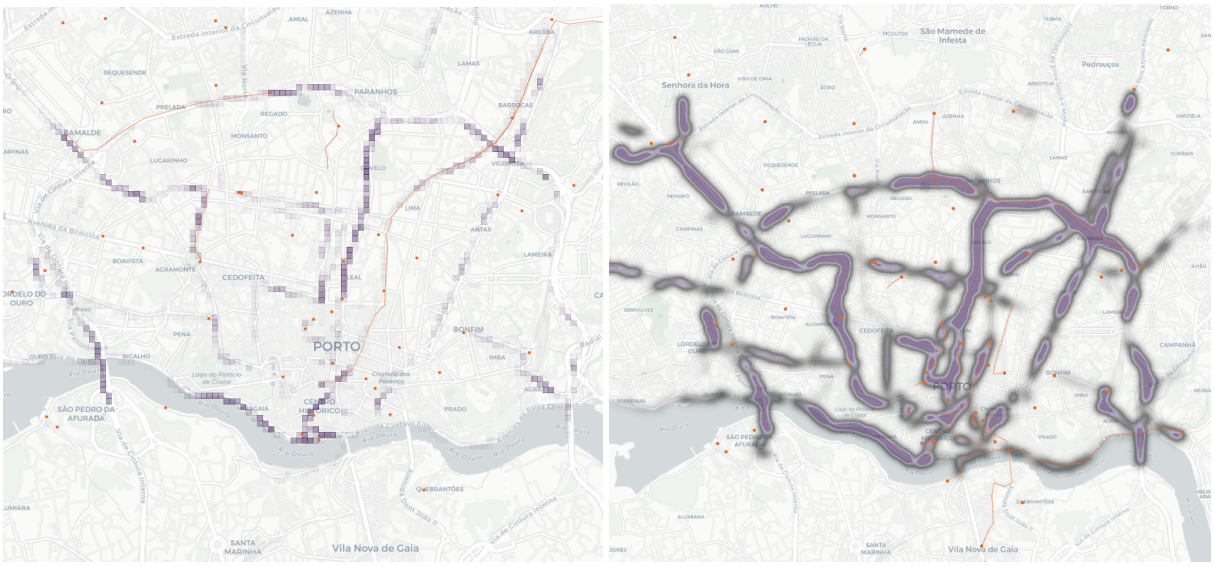


Figure 3.14: The history period's display methods.

Each representation has different strong points and tradeoffs. In the cluster heatmap, it is easier to understand the weight of individual points, but overall, it looks less visually clear, as the large squares do not represent what should be irregular-looking streets as well. The spatial heatmap is more clearly understood from a top-level view, as the more irregular shapes match city streets better, but the detail of individual squares and their attributes is lost. We performed user testing on both these representations

Figure 3.15: Different grid levels in the history period. As the user zooms out (left to right), the grid will decrease in resolution.

to conclude what users preferred and what was most effective (which we will discuss in Section 5.2).

In both representations, the different resolution grids are used in conjunction with map zoom (Figure 3.15): as the user zooms out, the visualization will show a lower-resolution version, while zooming in shows a higher-resolution version. This helps the grid detail accompany the map's overall detail (naturally, if a user zoomed out, they likely wish to see the overall trends and are not as interested in the details, while if they zoom in, they likely want to focus on detail).

3.4.4 Overview

The three periods are always drawn on-screen at the same time, the history period below the recent period, and the recent period below the ongoing period (Figure 3.16). We organize each period in this order because we want the more recent and higher-detail periods to stand out.

Figure 3.16: Visualization with the three time periods.

3.5 Summary

In this chapter, we presented the concepts for FastGeo. We discussed the concept of time periods, which allow us to separate time into three different periods and represent them in different ways. We then presented the trajectory bundling algorithm we use in the Recent period, which reduces the number of lines being shown by bundling similar lines into one. Afterward, we explained the grid binning algorithm, which we use in the History period to aggregate geospatial data into a set of grids. Finally, we presented the different representations used for each time period: animated raw lines on the ongoing period, trajectory bundled lines with color and width scales on the recent period, and spatial or cluster heatmaps for the grids in the history period.

Chapter 4

Prototype

To study the concepts we discussed above (time periods, overlapping visualization techniques in a map view, controlled simplification of data, and so on), we developed a prototype that would implement these concepts.

In this chapter, we present and discuss the resulting prototype, going through the implementation decisions we have taken throughout the development process to study the aforementioned concepts.

4.1 Architecture

The prototype's architecture (Figure 4.1) is centered around a `node.js`[1] server. This server manages the entire functionality of the prototype: it simulates data being streamed in and subsequently processed through each time period (something we will discuss further in Section 4.2). This requires calling processing functions that will interact with the database, as well as sending the data to the visualization.

A `PostgreSQL`[2] database with a `PostGIS`[3] extension is used, as it enables us to perform spatial queries on the data (such as the one used when filtering data for trajectory bundling), and automatically indexes spatial data, ensuring these queries are efficient.

There are different types of data processing involved in each period. We started one of the most important types of data processing (trajectory bundling) with Understand My Steps (Sil and Gonçalves, 2018), which was based on `Python`. So, for data processing, we decided to use it, as it is well suited

[1] `node.js`: `https://nodejs.org/`, last visited December 30th, 2020

[2] `PostgreSQL`: `https://www.postgresql.org/`, last visited December 30th, 2020

[3] `PostGIS`: `https://postgis.net/`, last visited December 30th, 2020

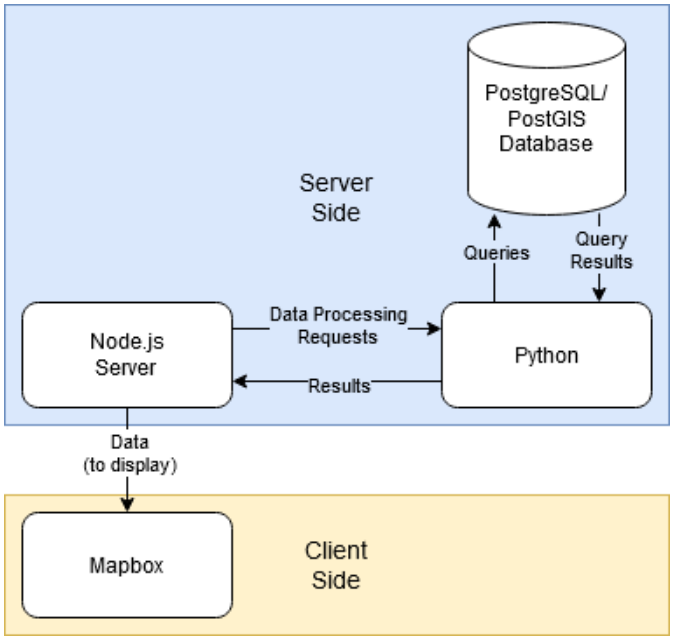


Figure 4.1: The prototype architecture.

to data processing, due to the wide variety of data processing libraries it has (such as `NumPy`[4]). We use `node.js` to create parallel `Python` child processes that can run on different CPU cores, greatly improving performance.

4.2 Data Processing and Simulation

While FastGeo is meant to handle streamed data, the prototype we have developed uses a static dataset. This dataset is processed through a simulation that allows it to appear as if streamed for us to build this prototype considering streamed data.

As we have previously discussed, GPS data is often inaccurate, and it requires processing to be properly visualized. Since our data is to be provided from a streaming source, we naturally cannot preprocess it in the way more static visualizations can. However, we still want to avoid the visual artifacts inaccurate data may result in.

[4] `NumPy`: `https://numpy.org/`, last visited December 30th, 2020

4.2.1 The Simulation Process

The simulation process starts by obtaining the data (for the dataset we used in evaluation, a CSV file) and inserting it into a database table (with no processing being done, naturally). Since we require no preprocessing at this point, no CSV parsing is done; we insert the CSV file into the database table. It should be noted that this process is easily adapted to any file format: the only aspect of our prototype that would require changes would be this part of the process, as once the data is inserted into this first database table, the format it was originally in is completely irrelevant.

After handling the aforementioned starting data transfer, the server creates the three time periods. Several variables of these time periods (such as the size of grid bins, how aggressive trajectory bundling is) will vary according to the user configuration. This is a configuration file where the user can adjust several variables that we will discuss throughout this chapter as they see fit. For each period, a `Python` child process is created, which will handle data processing for its period and only end when the main process ends. The server then starts two recurring processes that run in parallel.

One of these processes has an internal simulation time that starts at a user-defined timestamp. The process will constantly advance this time and fetch new data (that is, part of the dataset that has a timestamp between the simulation time updated in each step and the previous time) at a user-configurable pace (for example, the user can configure it so that one second of real-time consists of one minute of simulated time).

The amount of data that is fetched by this process depends not only on the pace but the real-time that the last fetch took as well (for example, if the last fetch took 0.3 seconds, then the server's simulated time will advance $0.3 \times pace$ seconds and fetch what it considers to be new data). The data is accumulated on a different database table to be then used by the other recurring process.

The other process will make every period update, receiving new data, and removing data that no longer complies with the period's constraints. All the accumulated data from the first process is used

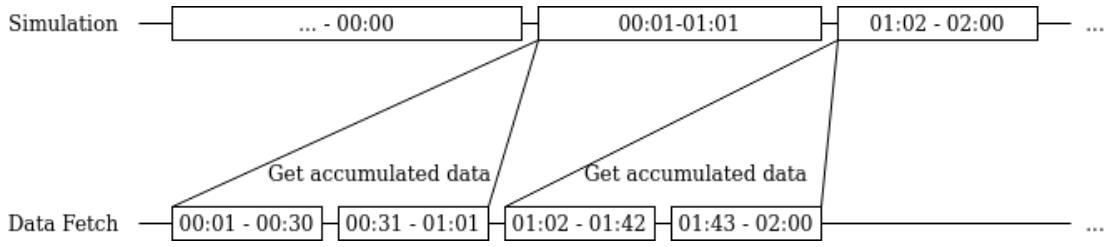


Figure 4.2: The prototype's data fetch/simulation processes and their interaction.

as new data here (for example, if one step of this process takes 1 second, then the following step will have $1 \times pace$ seconds accumulated, whereas if it takes 2 seconds, it will have $2 \times pace$ seconds). This two-process setup separates data fetching from data processing to ensure that the simulation is realistic enough since it is always receiving data at the same pace without depending on how long a simulation step takes.

Below, we will discuss the data updates time periods perform every simulation step.

4.2.2 Time Periods

The prototype always has ongoing, recent, and history periods. Every period consists of a corresponding table in the database, a `Python` child process, and methods to handle ingoing and outgoing data. Ingoing data comes from the initial select query in the ongoing period and the previous period's outgoing data in the recent and history periods. Outgoing data consists of lines that are removed when they meet each period's conditions for removal (this will vary depending on the period). As for the history period, since it is the final period, it does not produce outgoing data.

Finally, there is a method for displaying data that varies with each period. While displaying data is not strictly part of the data processing, there may be additional processing required for certain display methods. We will discuss these display methods further when we address the visualization itself.

These methods consist of communication with each period's child process. The `child_process` module from `node.js` allows for the creation of these processes and provides us with easy communication between the main module manager and each process. The communication consists of a simple message being sent. The child process interprets the message and performs processing based on its contents (removing, updating, or displaying data in different ways).

There is a module manager, which every simulation step calls the data removal method for each period, which lets them check their data and decide what can be removed for insertion in the following period. Afterward, the server will call the data update method for each period, allowing them to process whatever data they may have received after the removal method and insert it on their respective database tables. Finally, the server calls the data display method, letting each period return data to be displayed on the visualization.

These method calls are performed asynchronously to maximize their efficiency. For each round of method calls (removal/update/display), the manager waits for all calls to return before beginning the next round. Since these methods consist of communicating with each period's child process, they will only return when the child process returns a message stating it is processing is complete.

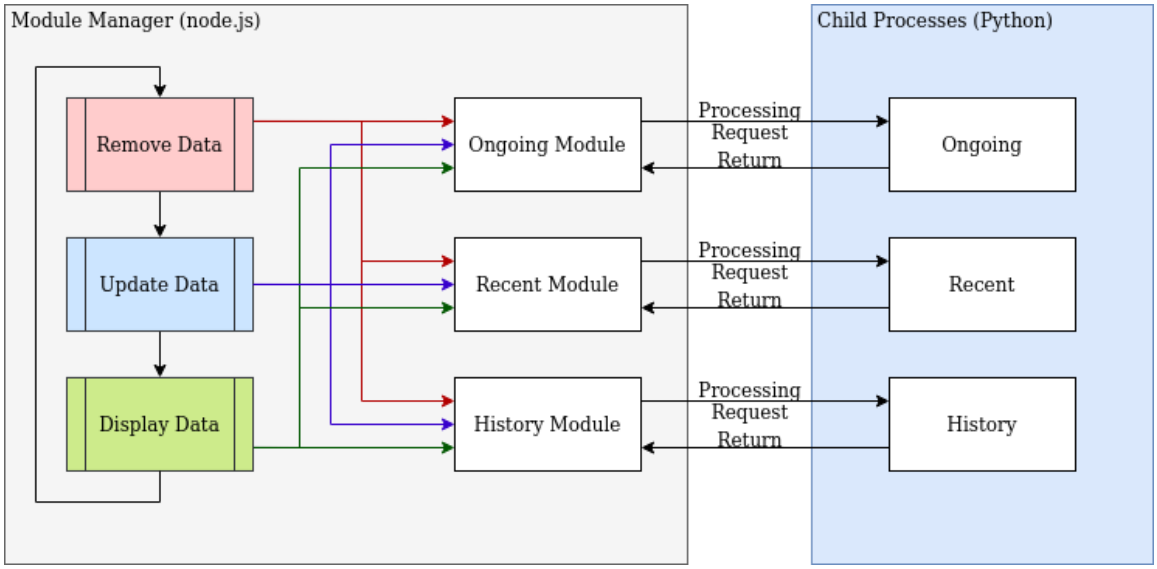


Figure 4.3: Diagram describing the simulation loop. The three parts of the loop (removal/update/display) call methods from the time period modules (colored lines), each of which requests processing from the corresponding `Python` child process and awaits the end of processing.

The three periods are represented by classes that inherit from the same abstract class. The use of polymorphism enables us to simply call the same method for every period independently of its type: this allows us to create an arbitrary number of different implementations for data removal and updating, so long as they follow this remove/update paradigm.

The ongoing period receives processing in the form of removal of unrealistically high-velocity sub-trajectories, that is, points where objects deviate from their trajectory in an unrealistic manner (often due to GPS errors), as these result in uncanny trajectories being displayed in a visualization. Apart from this, duplicate points are removed as well.

Ingoing data consists of the data selected from queries in each simulation from the unprocessed data table. After the aforementioned processing, it is put into this period's table. Outgoing data is selected by segmenting tracks by their stop events (we will discuss stop events further below). If the segmentation algorithm decides that the latest segment consists of a vehicle that is still in motion, then that segment is kept in the ongoing period. Otherwise, it is removed. This outgoing data will be sent to the recent period. This period's display method simply sends the data to the prototype's front end.

Ingoing data for the recent period consists of the outgoing data from the ongoing period. This data will be processed through trajectory bundling. Trajectory bundling simplifies tracks, avoiding having any

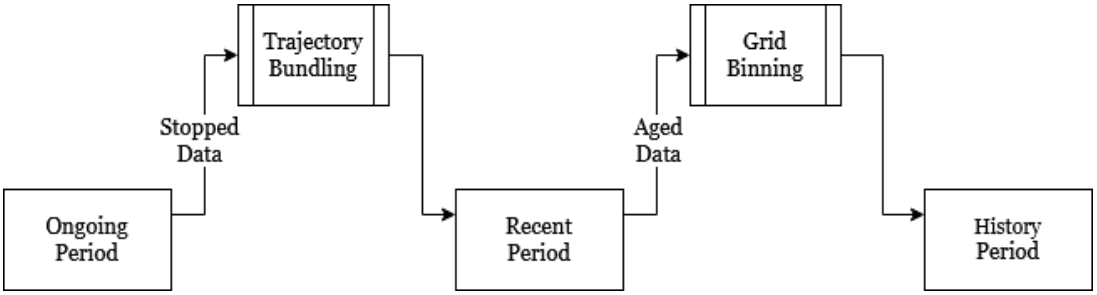


Figure 4.4: Data flow through the three time periods.

two parallel lines; the result is a network of lines that do not overlap, containing a number that indicates how many tracks are contained in each line.

Outgoing data consists of any temporal bins that have become older than the period's time limit. If only part of a segment's bins is eligible for removal, then the segment will stay in its current period and lose only its bins (resulting in a loss in overall track total). Otherwise, the entire segment is removed.

For the history period, ingoing data consists of bundled lines from the recent period that have aged more than the defined limit. These lines are processed by grid binning. For this period, there is not outgoing data, and the removal method is inexistent.

In each period, the display method returns the data to be displayed in `GeoJSON` format. This format is used for the lines, points, and squares stored in each period's data. The display methods only send new data and changes to the old to minimize the amount of data that is processed and sent. If there is a new line, point, or square that does not yet exist on the visualization side, it will be sent. Otherwise, we only send data that can identify a geometry for removal (for example, a line passing from ongoing to recent) or alteration (a square bin in the history period changing its total). This greatly reduces the data that is passed around from server to client.

The ongoing period relies on stop event detection to segment tracks and understand what needs to be removed and sent into the next period. Stop events are problematic in GPS data visualization because, due to the inherent errors in GPS, objects may be shown moving only a few meters while in reality, they are still, and that small movement may also result in strange artifacts around the object's stopping position.

TrackToTrip (Gil and Goncalves, 2016) uses DBSCAN clustering (Ester et al., 1996) with spatiotemporal features to detect and remove the artifacts stop events tend to produce. However, this is a solution only suited for a static visualization using a static dataset, where the programmer has the benefit of already having all the data with complete stop events. When data is streamed in, we cannot detect stop

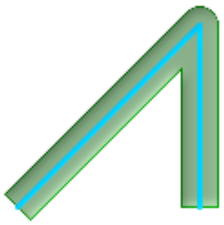


Figure 4.5: An example of the `ST_Buffer` PostGIS function. The blue line is used to create the green area.

events in this way, as we will not have the full stop event until the corresponding object begins moving again (and at that point, we would already be showing the aforementioned artifacts if no processing was done).

A simple way to detect stop events is checking if the velocity between two points is unrealistically low. If the object has only moved a few meters in the space of a minute or more, then this is likely to be a stop event if the data's update rate is low (otherwise, it may be a vehicle in a traffic jam, for example).

For simpler datasets, this is sufficient. Sometimes the update rate is so low (more than five minutes in many cases) that simpler measures are required for detection. If a track segment is older than a given threshold, we assume that the object stopped moving between the end of that segment and the beginning of the following one, and we treat it as a stop event. Finally, if a track has not been updated in some time (the exact time being configurable), we consider this a stop event as well.

The dataset we used to evaluate the prototype (more details on this dataset in Section 5.1) already contains information about stop events, so we take that information into account to handle the transition from ongoing into recent.

4.2.3 Trajectory Bundling

We've discussed trajectory bundling extensively in Section 3.2, but below, we'll specifically discuss several implementation decisions regarding our algorithm.

Regarding how we filter the data to compare, we use one of the functions in our database, `ST_Buffer`[5],

[5] `PostGIS ST_Buffer`: `https://postgis.net/docs/ST_Buffer.html`, last visited December 30th, 2020

which given a line, creates an area that represents all points whose distance from the line is less or equal to the given distance (Figure 4.5). The buffer size (that is, the given distance) is user-configurable.

For every new line, we perform one such query. The resulting selection of old data is given the new line's ID, marking it as selected by that line (if an old line is selected in several queries, then it will have several IDs). With this, we ensure that any line will only be compared with lines near it.

4.2.4 Grid Binning

While we've already discussed grid binning in Section 3.3, we did not discuss the implementation details.

Each grid is a sparse matrix, each integer on it being the total number of objects that passed through the corresponding square on the map. By using such a simple structure, we save a lot on both memory and processing power.

For example, if we use a grid with a resolution of 15 meters for an area of 10km by 10km and use 32-bit integers, the grid size is around 25MB. Since we are using `Python`, the number could be much higher, as Python integers are entire classes and naturally occupy more memory, but we can use `NumPy` integer types (e.g., `numpy.uint32`) to avoid this problem.

It is also simple to alter squares in this specific structure: we store the minimum and maximum (x, y) coordinates of the grid, therefore finding which square (i, j) in the grid matrix to alter given a point (x, y) is done by simply computing the geographic distance between the minimum (x_{min}, y_{min}) coordinates of the grid and the point:

$$i = distance((x_{min}, y_{min}), (x, y_{min}))/resolution$$

$$j = distance((x_{min}, y_{min}), (x_{min}, y))/resolution$$

At which point accessing the square is an O(1) process, as it simply consists of accessing index j of the array corresponding to index i of the matrix.

We use the above method to handle grid expansion. When expanding the grid, the previous grid is now only a subset of the new one, ranging from indexes (i_{pmin}, j_{pmin}) to (i_{pmax}, j_{pmax}) of the new grid. We compute (i_{pmin}, j_{pmin}) with this method and derive (i_{pmax}, j_{pmax}) from the result, allowing us to replace this submatrix of the new grid with the previous grid's matrix.

4.3 Visualization

To implement the visualization side of our prototype, we used HTML, CSS, and Javascript (with some libraries that we will discuss below) to create a web application that would show a map with several elements overlaid on top of it. Below, we will discuss decisions regarding the map and colors we used for the visualization and how the data that is processed on the server-side is used here.

4.3.1 Map

To display the map view, we use `Mapbox GL JS`[6]. This `JavaScript` library uses `WebGL` to provide maps and visualizations through a browser. We started by using `Leaflet.js`[7] and then `OpenLayers`[8], but we quickly came to the conclusion that despite the features both these libraries offered, their use of the HTML Canvas as opposed to a GPU-focused solution was a downside we could not accept. The number of objects on display made the map interaction slow down to the point of framerates in the single digits and sometimes even below this (less than one frame per second), which naturally made the visualization unusable.

Since `Mapbox` offloads the map and visualization display to the GPU, the performance is considerably better, making for better interaction (we will discuss framerate specifics in Section 5.1). The extra visualization features it offers (such as heatmaps) are done using GPU shaders, ensuring better performance than a CPU-based solution. Moreover, it allows future iterations to use custom shaders, possibly allowing more visualization techniques to be created with good performance.

The zoom level is an important factor. We set different zoom thresholds for each detail level of the history period's grid, meaning that, if a user zooms in beyond a certain threshold, the history period's representation will disappear and be replaced by a higher-resolution variant. Since we have three levels, this will happen twice.

`Mapbox` allows us to create expressions that consist of complex formulas (including several types of interpolation) using different attributes from the map or the data being displayed. We use these expressions to adjust certain attributes based on the zoom level dynamically. For example, we adjust the width of lines on the recent and ongoing periods based on zoom. This is done to keep line width stable relative to the width of streets: when zoomed out, lines will appear small enough not to occlude each other, while, as the user zooms in, they will become gradually wider so the user can analyze them

[6] `Mapbox GL JS`: `https://docs.mapbox.com/mapbox-gl-js/api/`, last visited December 30th, 2020
[7] `Leaflet.js`: `https://leafletjs.com/`, last visited December 30th, 2020
[8] `OpenLayers`: `https://openlayers.org/`, last visited December 30th, 2020

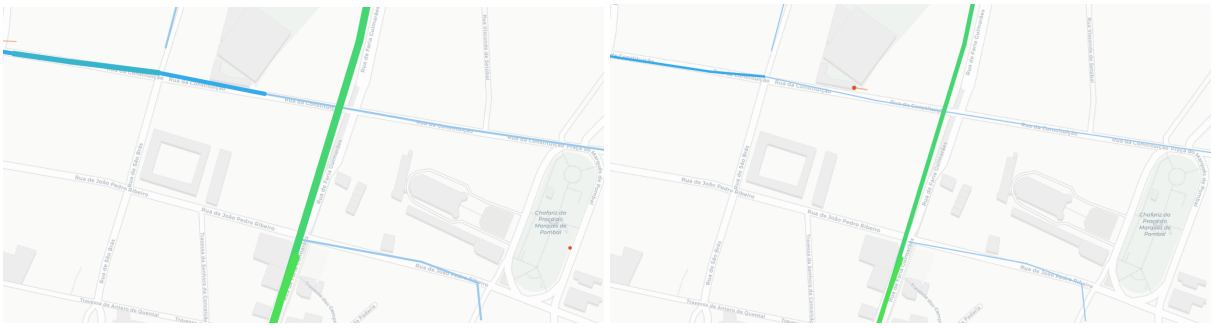


Figure 4.6: On the left, line width adjusted by zoom level. On the right, unadjusted line width. Width adjustment makes lines wider and somewhat easier to analyze when zoomed in.

in more detail (see Figure 4.6).

Zoom-related expressions are used in the history period's spatial heatmap as well: in this case, we want to keep the heatmap's points from looking like a set of disconnected points as opposed to a heatmap (see Figure 4.7), so we adjust each heatmap point's radius to avoid this from happening, keeping a stable heatmap independently of zoom. This relies on simple linear interpolation, but many interpolation points are required to get a stable heatmap at any zoom level.

Figure 4.7: Separated heatmap points. This is fixed by carefully setting up interpolation values based on zoom level.

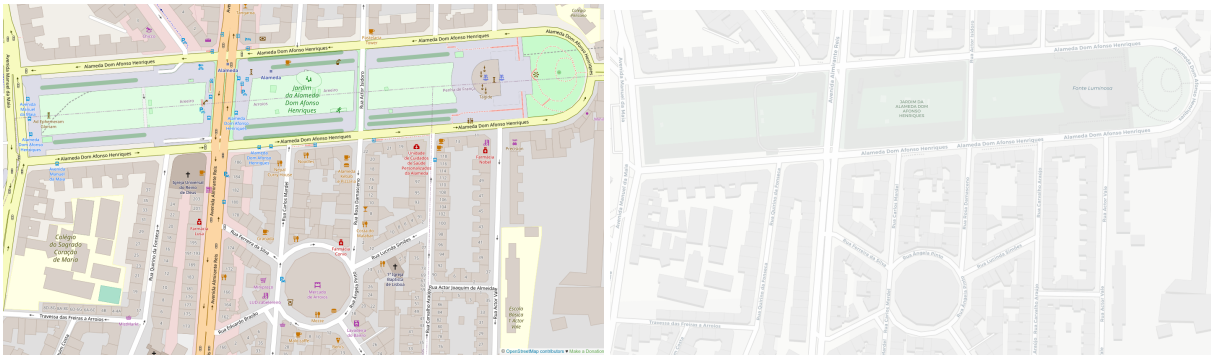


Figure 4.8: A comparison between the default `OpenStreetMap` style (left) and the `Carto Basemaps light_all` style.

4.3.2 Colors

For our map source, we decided to go with `OpenStreetMap`[9], as it is *open data* and allows free use. Since the default OSM style (Figure 4.8, left) is somewhat cluttered with all manner of signs (stoplights, bus stops, pharmacies, etc.), which despite being more informative to users ends up creating too much noise when overlayed with several visualizations, we decided to use `Carto Basemaps`[10] (Figure 4.8, right), which offer several less cluttered styles for free. More specifically, we used the `light_all` variant, which is a light style with mostly low-contrast white and grey colors, much more appropriate for a visualization that focuses heavily on color.

If we are going to use color to differentiate periods of time, then the colors we use must be easily differentiated. We started by using ColorBrewer[11], which is designed to help to select color schemes for maps, considering specific needs such as the nature of the data (Harrower and Brewer, 2003). When making a visualization focused on time, the first choice would be a sequential scale since time is naturally sequential. However, the way we have divided time in our visualization is by creating discrete periods and focusing on differentiating them. Therefore we went with a qualitative scale between the three periods (Figure 4.9).

Of course, we later went with a sequential color scale when representing the recent period. We decided to adapt the color scheme above to four colors. To do so, we used a color tetrad from the color scheme's orange (Figure 4.10). Color tetrads are two pairs of complementary colors, and the result is that these four colors have good contrast between them. For the sequential color scale, we used a

[9] `OpenStreetMap`: `https://www.openstreetmap.org/about`, last visited December 30th, 2020

[10] `Carto Basemaps`: `https://carto.com/location-data-services/basemaps/`, last visited December 30th, 2020

[11] ColorBrewer: `https://colorbrewer2.org`, last visited December 30th, 2020

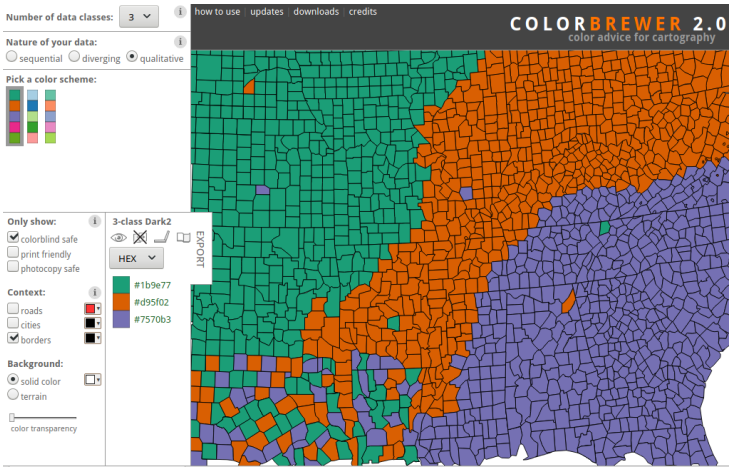


Figure 4.9: The color scale we started with when using ColorBrewer.

blue-green scale. The reasoning behind this was a process of elimination: scales using purple looked too dark, and a scale from green to orange has yellow in the middle, which makes it look like a diverging scale and not sequential.

Figure 4.10: The resulting color tetrad when using a similar orange to the ColorBrewer scheme.

To keep tracks with lower amounts of total from cluttering up the map too much, and to let the user focus on the greener hues, the final color scale starts with a less saturated blue, and subsequently transitions to the more saturated, darker blue and then to green (Figure 4.11).

Figure 4.11: The color scheme used in the recent period.

For the history period, color depends on the representation. For the cluster heatmap, we went with a simple scale from an almost transparent purple to the same purple with higher opacity (still not picking an especially high value of opacity, as we do not want the heatmap to obscure the map entirely).

Spatial heatmaps, however, are more complex when it comes to color. When it comes to this type

of heatmap, the color scales used often cover a lot of different hues (for example, VAUD (Chen et al., 2018), which we discussed in Chapter 2, has a heatmap with a scale from dark blue to red). Naturally, we have a limited space in hue to work here, but it was important to use several colors to communicate to the user that this is indeed a spatial heatmap. Having little range to work with in hue, we then focused on saturation and luminance. The resulting color scale uses purples and greys at different luminance to provide the visual effect of a spatial heatmap without using various hues (Figure 4.12). Like in the cluster heatmap, these colors have a relatively low opacity so as not to obscure the map.

Figure 4.12: The color scheme used in the spatial heatmap.

For the ongoing period, we used the remaining color in the tetrad, orange. To differentiate between points and lines, we made the point color slightly darker and redder (Figure 4.13, left), and show them at full opacity, where as lines have a slightly lighter color, and their opacity is lower (Figure 4.13, right).

Figure 4.13: Colors for ongoing points (left) and lines (right).

4.3.3 Data

When a simulation step happens, the display data resulting from each period's display method is sent to the visualization. This data includes new data points from each period, as well as adjustments to make in the visualization's internal data for it to remain consistent with the simulation. For example, lines that have passed on to a new period and have to be removed in the visualization, or certain grid squares that have a new number of segments associated with them. These removals and updates are done, and afterward, new data is added. The nature of this data is, of course, reliant on the display method. While all of the methods receive a set of GeoJSON features, the feature type depends on the method: for heatmaps and the ongoing points, we use `Point` types, for line-based methods, we use `Line` types, and for the grid squares we use `Polygon` types.

After updating the data, we perform some additional processing. We change the color scales to fit

with the maximum total in the current data for the recent and history periods. This is a matter of changing the corresponding `Mapbox` expressions. The expressions for these scales consist of linear interpolation between several values of total to return a color.

Aside from changing color scales, ongoing lines and points that have happened in the current simulation step are animated. This animation consists of showing a gradually greater fraction of each line every frame, using interpolation. The interpolation factor starts at zero and gradually increases with each frame (and, of course, the rate of growth is based on time passed and not frames passed, avoiding problems such as animation speed depending on framerate), eventually reaching 1, which corresponds to the complete segments being shown and the end of the animation. In cases where the animated lines have more than one segment, the interpolation algorithm decides which segments to show complete, which segment to show interpolated, and which segments not to show. For example, a line with three segments and a factor of 0.5 results in the first segment being fully shown, as well as half the second segment.

4.4 Summary

After discussing the concepts in Chapter 3, we discussed how they were implemented in a prototype. We started by presenting the prototype architecture, discussing each time period and their manager's structure, and the technologies we used for each part of the architecture. We then discussed how data processing, such as the algorithms for trajectory bundling and grid binning, are implemented. Finally, we discussed implementation choices for the visualization, such as mapping libraries, map providers, color scales, and data handling.

Chapter 5

Evaluation

In this chapter, we present how we tested the prototype regarding both performance and usability. In Section 5.1, we detail the performance tests we made and draw comparisons between different versions of the prototype and discuss its limitations. In Section 5.2, we describe how the prototype's usability was tested and present the test results.

5.1 Performance

When making performance tests to our prototype, our goal was to understand the prototype's limitations concerning how much data it can process. Essentially, we wanted to find what amount of data is too much for the prototype to handle. We analyze two main facets during this section: the server-side performance, that is, how fast it can process data, and the framerate on the client-side, that is, how fluid the visualization looks to the user.

5.1.1 Methodology

To test the prototype's performance, we used Mozilla Firefox (83.0, 64 bits) on Windows 10, running on a laptop with an `Intel Core i7-8750H (2.20GHz)` CPU, 16GB of RAM, and an `NVIDIA GeForce GTX 1060` GPU, with the browser running on a 1920x1080 window.

We used a dataset with taxis in Porto, Portugal [1] from Moreira-Matias et al. (2013)).

[1] Porto Dataset: `https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i`, last visited December 30th, 2020

To evaluate the frame rate in the map view, we used a frame rate counter plugin for Mapbox [2], and let the visualization run for twenty seconds when each period already had data being displayed. While testing, we did not perform any specific interaction: we simply let the visualization run.

To compare our optimized trajectory bundling algorithm with the original, we ran the Porto dataset for 80 simulation steps on both versions of the algorithm. In the results we present, we are only comparing the time it takes both algorithms to perform trajectory bundling on the Recent period.

For the overall prototype performance, we started by running the simulation for a full 24 hours of simulated time, at a realistic pace (that is, 1 second of real-time is 1 second of simulated time). Due to the way data fetching works in the prototype, if a simulation step takes more than 1 second, it will process over 1 second of data. To analyze how simulation step time evolves, we ran the simulation at each hour for 20 steps and averaged the step duration.

5.1.2 Map View

When interacting with a visualization, users need to be able to do so with fluidity. If the visualization's frame rate is sub-par, for example, below 20 frames per second (FPS), a user may feel frustrated at, e.g., panning around the map or zooming, and feeling frustrated at how sluggish the interaction feels. Since our prototype uses a WebGL-based mapping library, it can take full advantage of the GPU.

Looking at the frame rate graphs (Figure 5.1), we can see that the frame rate maintains close to 60FPS most of the time, periodically having moments where framerate diminishes. These dips are

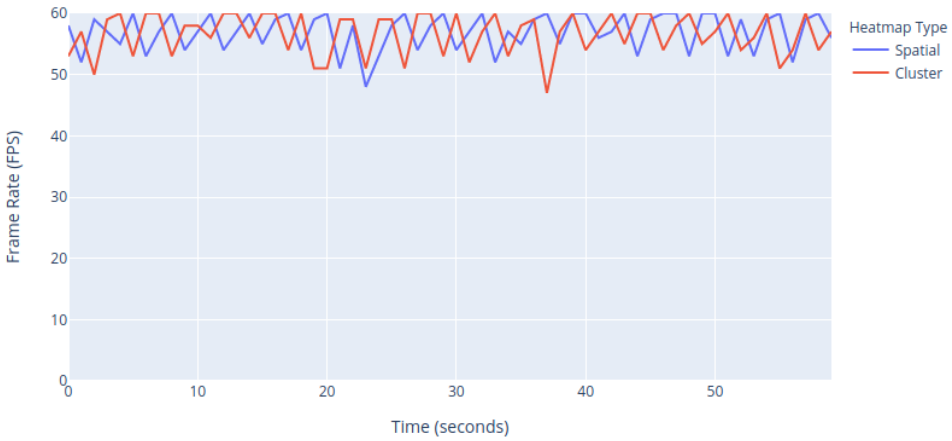


Figure 5.1: Framerate graph for both History period representations.

[2] `mapbox-gl-framerate`: `https://github.com/mapbox/mapbox-gl-framerate`, last visited December 30th, 2020

somewhat larger with the cluster heatmap representation for the history period, and they correspond to a short, periodic stutter when the visualization is receiving new data and processing the animations.

Essentially, when it receives data, the bottleneck becomes the CPU and not the GPU: the CPU becomes busy with updating the visualization and cannot produce enough frames to keep the frame rate stable. When interacting with the visualization, it runs at a stable near-60 FPS, but these stutters come across to the user as very short hitches when panning or zooming. These stutters are slightly longer when using the cluster heatmap representation due to the larger volume of data received: The spatial heatmap data is a set of points (one coordinate per grid bin), while the cluster data is a set of squares (four coordinates for grid bin).

We can conclude that the visualization maintains a mostly stable framerate, with the occasional stutter. While these stutters detract slightly from the interaction's fluidity, the frame rate is otherwise extremely smooth.

5.1.3 Trajectory Bundling

Before discussing the overall performance with regard to the streaming simulation and data processing, we will first focus on trajectory bundling. As we have already discussed, the algorithm we are using is based on the algorithm used in Understand My Steps (Sil and Gonçalves, 2018), but heavily altered to suit real-time streamed data.

Understand My Steps compares each segment to each other, resulting in $O(n^2)$ performance. Naturally, running this algorithm every few seconds with a gradually accumulating dataset is not feasible.

The optimized algorithm in our prototype does perform at close to $O(n^2)$, but only for data that has been added to the recent period in the latest simulation step (new data); as we explained in Section 3.2.1.1, data that was already in this time period (old data) is filtered in a way that greatly reduces comparisons, as new data will only compare with old data that is near enough.

For both cases, we also filter angles that are too different (hence the "close to $O(n^2)$", as this angle comparison makes it somewhat faster than $O(n^2)$). Essentially, many steps are taken to reduce the amount of comparisons as much as possible, as that is the most demanding part of the algorithm.

To give an example of how reduced the number of comparisons is, let's say that the recent period has 1000 segments, and in a simulation step, another 10 segments are inserted. Using Understand My Steps, we would have to compare each segment to every other segment, resulting in $1010^2 = 1\ 020\ 100$ comparisons.

Now, let's consider our algorithm with the same data. If given the 10 new segments, 5 are horizontal

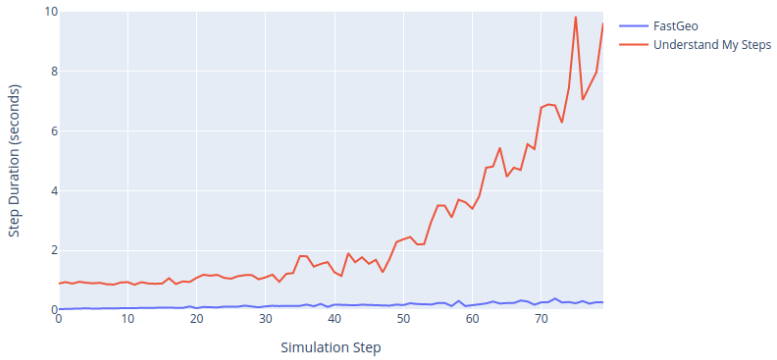


Figure 5.2: Comparison of bundling time between Understand My Steps and our optimized algorithm.

and 5 vertical, and if each of these segments is near and has a similar angle to 5 other old segments, we will compare the 5 horizontal segments with each other (since they have similar angles), the 5 vertical segments with each other. Then each segment will be compared to 5 others, the result being $(5^2 \times 2) + (1 \times 5 \times 10) = 100$ comparisons, that is, roughly 0.01% of the comparisons, or a $10\ 201$ times reduction.

Naturally, the reduction is likely to vary a lot, depending on how many segments are incoming in each simulation step and how different the angles are between them (since efficiency for these is still close to $O(n^2)$), and how many old segments match each new segment (here, efficiency is $O(n)$ for each segment).

The comparison process is greatly simplified, as well. We reduced the comparison from the 12 different branches depending on several variables of Understand My Steps to 2 branches depending on a single variable. Overall, we have 7 conditions against 20, which results in less time spent understanding whether two segments are eligible for matching.

We decided to evaluate our improvements by running the simulation with each algorithm for 40 minutes. The results (Figure 5.2) make it easy to conclude that our improvements greatly optimized the Understand My Steps algorithm, as both time and comparisons made are both much less and grow much less over time as well.

5.1.4 Simulation and Data Processing

After evaluating how improved the trajectory bundling algorithm was, we evaluated the overall performance. We would ideally like to keep each simulation step below 5 seconds. Therefore, if a step

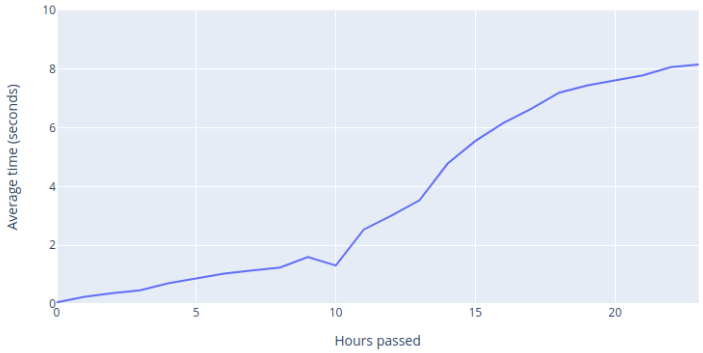


Figure 5.3: Average time for 20 simulation steps at each hour.

takes more than that, the next step will only process the next 5 seconds. This helps reduce the impact of spikes in performance. For example, if a step takes 10 seconds due to a higher number of trajectories passing from ongoing to recent in that interval, then not limiting the size of the next step's fetched data may lead it slowly accumulating more data it needs to process, and performance getting out of control. This limit often helps the simulation maintain steady times despite the occasional spike.

The results are shown in Figure 5.3. As we can see, the duration rises over the 5 second goal as time passes. The vast majority of this time is spent on the recent period, performing trajectory bundling. If the prototype only consisted of an ongoing period and a grid binning-based history period, each step would never go above 0.5 seconds, except for a few spikes that correspond to grid expansion, which always eventually stabilizes for most real-world cases that do not consist of extremely varied routes being taken (that is, if the use case consists of something like our dataset, focused on a city, then the grid will not need expansion after a certain point).

Regarding the recent period in specific, its slow performance is due not to the algorithm we are using, but to a factor associated with it: database usage. As the number of segments in the recent period increases, the overhead associated with the PostGIS database does so too. Since we have to transfer data around, we get this overhead that, while not exponential or quadratic in growth, nevertheless ends up growing too much for our goals. While using PostGIS to reduce the number of comparisons we make between segments results in greatly improved performance, the cost is this overhead, which proves to hinder our performance goal.

This means that the most limiting factor is how long the recent period spans, as a longer interval means that the database table will be larger. For these tests, we configured the recent period to consist of 2 hours (that is, from the simulation's "current time" to 2 hours before that time).

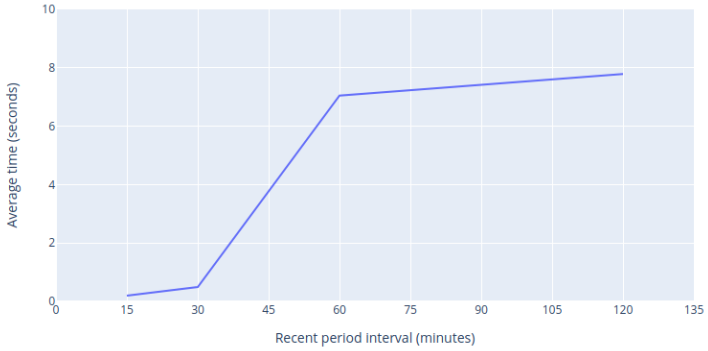


Figure 5.4: Average time for 20 simulation steps at the most computationally expensive hour, each average being a different interval of the recent period.

Comparing different intervals with the hour with the most traffic (Figure 5.4), we can see that a smaller interval for the recent period results in a much-improved performance.

Despite all the optimizations that have been made to the trajectory bundling algorithm, its performance is somewhat short of meeting our goals. Only if the recent period consisted of a very short interval is the algorithm fast enough to handle incoming data without too much slowdown. As for the other two periods, their performance is more than fast enough to meet our goals.

5.2 Usability

When evaluating the prototype's usability, our goal was to observe how users interact with it, that is if they were able to understand the difference between each data representation, if they were able to identify trends, and if they were able to understand how trends changed over time. Since we created two representations for the History period, we wanted to compare them and conclude which representation was better suited to represent this period.

We performed usability tests with users who were invited to test the prototype. The tests consisted of performing several tasks, most of them consisting of observing the prototype with the dataset at different times and answering simple questions with relation to different time periods, different parts of the map, or different times.

Before each test session, we presented users with a questionnaire (see Appendix A). The first section in this questionnaire presented a consent form, which users were asked to read and then verbally consent to. After the consent form, some questions regarding user characterization: gender, age, and a

short color blindness test (where the third image was just a control test). We made this short color blindness test instead of just asking if the user was color blind, as we could not be sure if users were aware of their color blindness or lack of it. After these questions, users were introduced to each time period's representation and given five minutes to explore the prototype for themselves and ask any questions.

After performing each task, users were asked to answer several questions presented in a Likert scale (Likert, 1932) regarding the different time periods and the two representations for the History period. Afterward, each user took the *Raw* NASA Task Load Index questionnaire. NASA-TLX is used to evaluate a task's workload, and the original test consists of two phases: rating each of the six subscales (Mental Demand, Physical Demand, Temporal Demand, Performance, Frustration, and Effort) and then performing pairwise comparisons to weight each subscale. The *Raw* version of the test we used eschews the weighting phase. We used this version because it is faster to apply and has evidence that proves it to have experimental validity (Bustamante and Spain, 2008).

Finally, users took the System Usability Scale (SUS), a series of questions that let us produce a value for the prototype's usability.

5.2.1 Tasks

When considering what tasks to require of users, we wanted the full set of tasks to have a good variety, comprising the evaluation of how users understand each individual period and the combination of them on a map that can change over time. We wanted to evaluate how users understood space and time as it changes on this prototype.

We started by creating a basic task for each period (ongoing, recent, history) to make sure users understood each period's basic representation (for the history period, this task would have to be made twice, once for each of the heatmap types). Afterward, we created more complex tasks requiring the user to compare different periods and compare the trends at different times or different parts of the map. Finally, we created two less constrained tasks to enable a more free-form analysis.

The final tasks, therefore, are:

1. Follow a specific vehicle and point out when its trajectory has transitioned to the recent period.
2. Indicate the most-used street recently.
3. Indicate the area that has seen the most traffic overall (this is, not just recently).
4. Explain whether, recently, there has been more traffic outside or inside the city (we defined inside/outside the city as being South/North of Senhora da Hora).

5. (After showing the prototype at two different times in the simulation,) explain whether trends in traffic have changed between these two times or not.
6. Indicate an area that was used in the past, but not recently.
7. Go to Ponte da Arrábida (if the user did not know where this was, we explained it was the western-most bridge in Porto) and indicate how many vehicles passed through it recently.
8. Indicate one vehicle that does not follow the trends displayed by the visualization.
9. A scenario will be shown. This scenario will have a gradual but significant change. When it happens, indicate it.
10. Explore the visualization and make any observations.

We selected a specific timestamp for each task that would avoid ambiguous answers and used the same timestamp in all tests so that the correct answer was always the same. Task 3 and 6, which focus specifically on the history period, were performed twice (once for each representation) to compare both representations' clarity and understandability (naturally, for each representation, a different timestamp was selected to have different answers in each).

To avoid the order of questions possibly impacting statistics (for example, task 1 being negatively affected by users' lack of familiarity with the prototype), each user performed the tasks in a different order. This order was decided by Latin squares, which, given n tasks, present n rows and n columns of different orders to perform them.

Of course, using this method would very likely affect the statistics in another way: if users get task 9 as their first, they would be likely to have greater difficulty than usual due to not having any other tasks to ease them into this one. Therefore, we decided to divide tasks into three sets with incremental difficulty, the order of tasks within each set following a Latin Square design: the tasks start with an order of tasks 1 through 3, then another order of tasks 4 through 8, then task 9 and finally task 10. We put task 10 last instead of randomizing the order of 9 and 10 simply because task 10 relies on free observation and letting the user explain trends he noticed throughout the entire testing process and explore the map freely.

For Task 9, the scenario used was running the Porto dataset from 4:00 to 5:30 (a.m.). At around 4:45, several taxis start moving out of town into the airport, resulting in several ongoing points in the airport, and lines in the recent period from the city towards the airport appearing (see Figure 5.5).

Task 10 is a freeform task where the user is asked to make any observations about anything they desire. In this task, we did not impose a time minimum or maximum or a correct answer. The users were

simply asked to voice their thoughts about whatever parts of the map or trends they noticed until they felt they had no further observations to present.

For each task except 10 (due to its freeform nature), we asked users to rate the difficulty from 1 (very difficult) to 5 (very easy). We recorded the time it took each user to perform each task as well. Additionally, we recorded the success or failure in each task's completion. As we mentioned above, each task was performed on a specific timestamp that would only have one correct answer, resulting in success.

5.2.2 Participants

A total of 21 people participated in the tests, resulting in 42 tests for Tasks 3 and 6 (considering we did these tasks for both representations of the history period). Of these participants, 13 were male, and 8 female and only three were not between 18 and 25 years old.

As for visual problems, we only retrieved data regarding participants' color blindness, as the use of glasses corrects other vision problems; color blindness, however, cannot be corrected and might impact the test results. None of the participants we tested were colorblind.

The tests were performed remotely, using Zoom to share the screen and let participants control the prototype remotely.

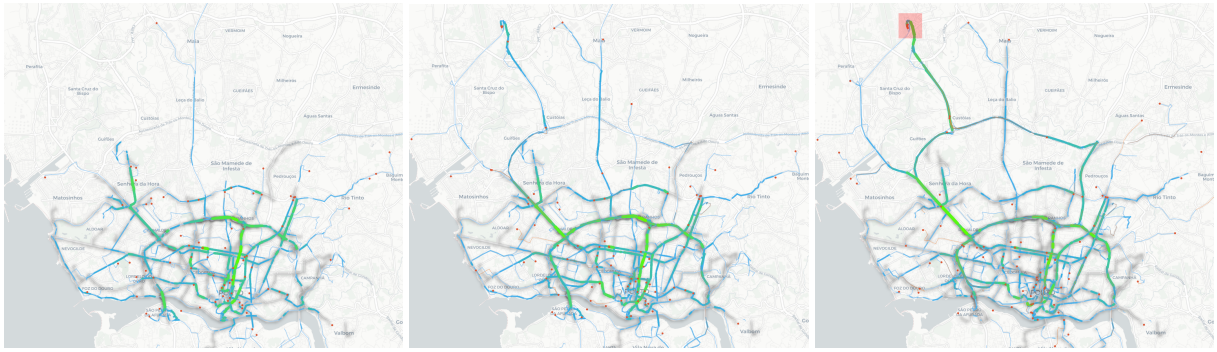


Figure 5.5: Temporal progression in Task 9's scenario. From left to right: 4:00, 5:00, 5:30. The airport is marked in red, in the rightmost image.

5.2.3 Results

Below, we will present the statistical results of the user testing process and the statistical analysis we performed on the results. For each graph, Tasks 3/C and 6/C correspond to Tasks 3 and 6 with the cluster heatmap representation of the history period, while 3/S and 6/S correspond to the same tasks with the spatial heatmap representation.

Our statistical analysis consisted of finding any significant differences between the two different representations and between each task. We evaluated, for each task, the completion time, success, and each user's reported difficulty. Additionally, we used questions we asked the users after they performed tasks.

We started by considering the null hypothesis (H_0), that is, that the differences between tasks and between different History period representations have no significant difference between each other. We will use different methods for different statistics, but they will all confirm or reject this hypothesis, using a reference p-value of 0.05.

To find significant differences between the two representations, we considered different methods. To analyze the time and difficulty of both tasks that have one variant for each representation, we first checked if they were normally distributed using the Shapiro-Wilk test (Shapiro and Wilk, 1965). None of the variables were normally distributed, the data consisted of pairs of samples (as we only have two different representations), and the samples were not independent (as the same people performed the tasks), therefore we decided to use the Wilcoxon signed-rank test (Wilcoxon, 1945). For test success, which is a dichotomous variable (success/failure), we used McNemar's test (McNemar, 1947). For task 6, both task variations had a 100% success rate (see Figure 5.6). Therefore we did not use tests and concluded there was no significant difference in this variable.

As we can see in Table 5.1, **the only significant difference between both representations was**

Task	Variable	Z	p-value
3	Time	-1.199	0.23
	Difficulty	-1.995	0.046
	Success	-1.134	0.453
6	Time	-1.148	0.251
	Difficulty	-1.897	0.058

Table 5.1: p-values for History period representation comparisons.

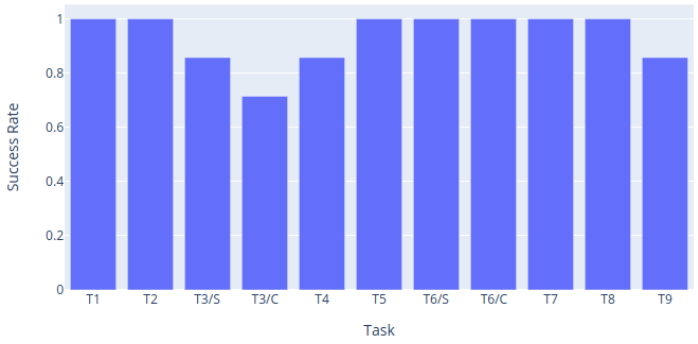


Figure 5.6: Success rate for each task.

found in the difficulty of Task 3, though the difficulty of Task 6 has a p-value somewhat close to the rejection point of 0.5 as well. Of course, this is only when considering task-specific variables: after the tasks were completed, users were asked to rate each representation, the transitions between periods, and how easily they felt they could distinguish each period with each representation of the History period, on a Likert scale. Once again, using the Wilcoxson signed-rank test for the questions specific to the History period, we found the differences between representations to be significant (with p-values of 0.003, 0.018, and 0.013, respectively).

The spatial heatmap representation was the most successful overall: comparing the statistically significant difficulties in Task 3, more specifically, their means with a confidence interval of 95%, the spatial heatmap variant of the task has values between 3.96 and 4.42, while the cluster heatmap variant has values between 3.39 and 4.04.

To see if we could find significant differences between each task (regardless of representation), we started by once again using the Shapiro-Wilk test to check if the variables followed a normal distribution. Since most of them did not, and the samples were once again not independent (as the tasks were always performed by the same group of people), we performed the Friedman test (Friedman, 1937). In case this test indicated significant differences, we performed post-hoc tests, more specifically Wilcoxon tests on pairwise comparisons, with Bonferroni corrections (Bonferroni, 1936).

In both time and difficulty, the Friedman tests indicated statistical differences (time: $\chi^2(2) = 179.588$, $p = 0$; difficulty: $\chi^2(2) = 86.923$, $p = 0$). We then performed post-hoc pairwise comparisons, leading us to several conclusions.

The most noticeable conclusion was that the time for Task 1 was significantly different from all the other times (all p-values were 0, the Z values are in Appendix C). This is due to a specific detail of

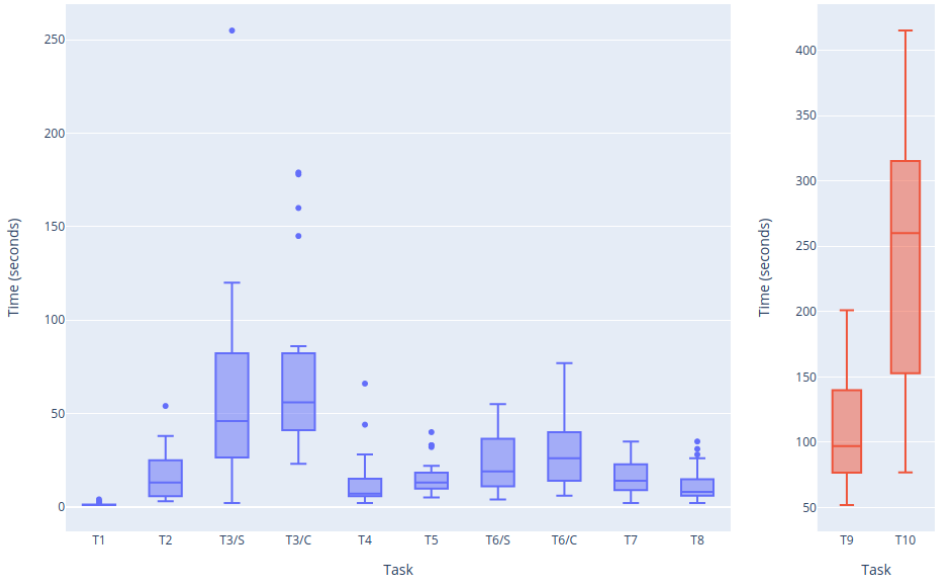


Figure 5.7: Time distribution for each task.

measuring time in Task 1: since this task consists of understanding whether the user noticed that a track changed from Ongoing to Recent, the time data for this task consists of how long it took the user to notice the change. All users noticed it almost immediately (the variance in time consisting of different Zoom latencies and/or reaction times), and therefore this time is minimal on average compared to the other tasks.

As for the other tasks, **Task 10's time is significantly different from all others** as well (all p- values were 0 except for the comparison with Task 9, which was 0.001). The difference is on a different end of the spectrum compared to Task 1 however: Task 10 is by far the longest on average, with values between 3:06 and 4:43 minutes. **Other tasks with several significant differences are both variations of Task 3 and Task 9**. Apart from Task 10, these are the longest, from Task 9 (with values between 1:19 and 2:12 minutes) to Task 3 (with the cluster heatmap variant having values between 0:53 and 1:37 minutes, the spatial heatmap variant between 0:37 and 1:27 minutes).

Looking at significant differences in difficulty, the pairwise comparisons indicate that they happen between two groups: Tasks 1, 2, 7, and 8, and Tasks 3 (both variants) and 9. These two groups are different extremes in difficulty: the first corresponds to the generally easier tasks (none of their values are below 4.63), and the second corresponds to generally harder tasks (none of their values are above 4.27, and the minimum lower bound is 3.39 in the cluster variant of Task 3).

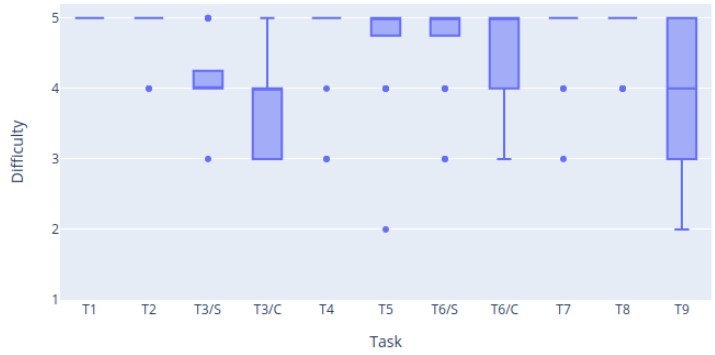


Figure 5.8: Difficulty distribution for each task. For this attribute, 1 was considered ”very difficult”, and 5 ”very easy”.

After the tasks were performed, participants rated how well they understood each time period in a Likert Scale. We used the Friedman test on the results and concluded there were significant differences ($p = 0$). After performing pairwise Wilcoxon tests with Bonferroni corrections, the cluster heatmap representation had a significant difference to every other representation/period, as it was the lowest-rated one for this question.

Overall, the most challenging tasks were Task 3 (both variants) and 9, and the most time-consuming consisted of the same group as well, with the sole addition of Task 10. **Considering the different representations in the History period, the easiest to understand was the spatial heatmap variant**, as its related tasks were generally both easier and less time-consuming. **The cluster heatmap variant was generally harder to understand**, and its related tasks more challenging.

5.2.3.1 Questionnaires

After the tasks were performed, we asked users to take the System Usability Scale, a set of ten questions that allow us to measure the usability of the prototype (the results are in Figure 5.9). Having a score above 87.5 would correspond to the top fourth of responses (Bangor et al., 2008). **The average SUS score for the prototype was 90.12, which would correspond to an adjective rating (Bangor et al., 2009) of *Excellent***, and 0.78 points below a rating of *Best Imaginable*.

Regarding individual questions, the questions with the most positive ratings were the fifth (how well integrated the system’s functions were) and the eighth (how complicated the system was to use). The question with the most varied answers was the fourth (whether the user would require a technician’s help to use the system).

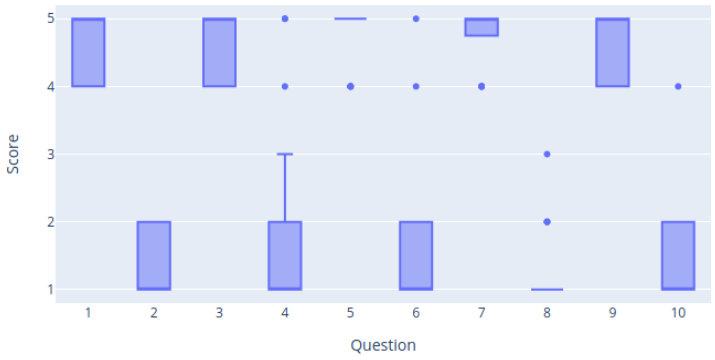


Figure 5.9: SUS distribution for each question.

Aside from SUS, users were asked to take the NASA Task Load Index as well. This test consists of six sub-scales that measure different facets of a task, overall letting users rate the perceived workload. The six sub-scales are used to calculate a final workload score: our result was 22.58.

Analyzing the complete results (Figure 5.10), we can see that users felt they performed the tasks well, as the Performance sub-scale has a median of 3 (in NASA-TLX, this scale represents Success or good performance with 1, and Failure or poor performance with 20), and a score of 12.86. Frustration was low as well, scoring 16.67, and Physical Demand was naturally low too. The highest demand was, naturally, Mental Demand, scoring 32.86. Finally, Temporal Demand and Effort scored 24.29 and 30.24, respectively.

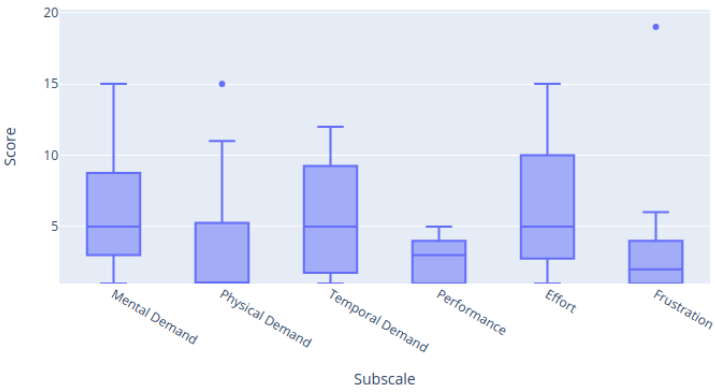


Figure 5.10: NASA-TLX score distribution for each sub-scale.

5.2.4 Observations and Feedback

Apart from statistical data, the user testing process provided us with many observations regarding how users interacted with the prototype, as well as their opinions and feedback.

While the statistic analysis we performed shows a clear preference for the spatial heatmap representation, it is not unanimous. Many people preferred the cluster heatmap. The preference came down to whether a user preferred the more immediately understandable differences of the spatial heatmap (due to it using three different colors and forming easily recognizable dark purple spots in areas with the most traffic historically) or the higher quantity of information the cluster heatmap provides (since it lets users hover over each square and get the exact number of objects that passed through it). Some users noted that the color scale used for the cluster heatmap did not have enough color range, making it harder to tell the higher total between two squares.

Even given the higher preference for the spatial heatmap, the difference in time (while not statistically significant according to the tests we performed) reflects a certain difference in how users interact with each representation. In the cluster heatmap tasks, users generally took more time because they tended to analyze each square individually. Some users requested that this hover feature was added to the spatial heatmap as well.

Task 10, as we have discussed, was mostly open-ended, allowing users to explore the prototype freely and pointing out anything they wished. There were many different observations that users posed, but a frequent one was curiosity regarding the airport. Users commented that cars were probably entering the city until they noticed the airport. Eighteen of them showed curiosity (five of them taking correct guesses at what it was), and understanding that it was an airport seemed to re-contextualize people's observations, as they then understood that the high traffic outside the city consisted of taxis likely picking up people from the airport.

This is an interesting observation, as it shows more than a simple understanding of the visualization: rather than only understanding the base spatiotemporal context (e.g., *"there are many vehicles in this part of the map and a lot of them passed recently through these different parts of the map"*), they also understand a more complex context, thinking not only of parts of time and space with no connection between them, but thinking of this data as the real traffic patterns of a city, and connecting these separate observations to form conclusions about how traffic flows throughout the city over time (e.g., *"at this point in time there is not much traffic inside the city - only on the History period - because most taxis are busy picking up people at the airport, and eventually they will start to get back to the city center and recent traffic will increase there"*).

Users suggested that the map showed some points of interest (such as the aforementioned airport) and a feature that let them filter out a period and analyze it individually. While some users were pleased with the color choices, others suggested using more colors for the color scales, specifically for the History period (different users would suggest this for different representations).

5.3 Discussion

Given the results we obtained during the user testing process, questionnaires, and general feedback, we can conclude that users were generally able to understand how the prototype worked, the differences between each period, and how trends could evolve over time. Our goal was to make a visually clear and easily understandable way to show trajectory data, and users were not confused with the representations we created for our prototype.

Given that the Ongoing period is the most simple, it is no surprise that users understood it very well. Tasks 1 and 8 focus on this period and they had excellent results. Users had no difficulty understanding the Recent period: tasks 2, 4, 5, 6, and 7 were focused on this period, and all of them had good results. The History period was somewhat harder to understand, with tasks 3 and 6 focusing on it and having somewhat inferior results compared to the other two periods.

Performing statistical analysis to compare the different representations in the History period, we can conclude that the **spatial heatmap** was both generally preferred by the users and easier to use, with better results regarding difficulty. Some users did prefer the **cluster heatmap** representation, but they were a minority.

Users tended to have less positive feelings towards the cluster heatmap due to its more complex appearance (many discrete squares as opposed to a continuous surface) and its color scale (which had too little range in color). Changing the cluster heatmap's color scale could improve this visualization's usability. As for the spatial heatmap, a hover tooltip similar to the cluster heatmap could be added (which was not done due to limitations related to the Mapbox library).

Users had some more difficulty analyzing how trends can change over time: While task 5 had very good results, the more complex task 9 proved to have significant differences compared to most other tasks. However, the results were still perfectly satisfactory, given that task 9 is the most complex.

Task 10 was focused on letting users observe the visualization freely. Their observations provided us with good information regarding how they interpreted the visualization, comparing the three different periods to conclude the trends being shown.

Considering the SUS and NASA-TLX scores, we can conclude that the prototype's usability was

overall excellent. Tasks such as 9 could have had better results, but we must keep in mind that this is a visualization made for analyzing large amounts of data as it evolves in real-time, which is a complex subject to tackle.

Regarding performance, the visualization's performance was acceptable, only having small stutters when updating data and keeping a steady 60 FPS when not doing so. While this aspect could naturally be improved by further optimizing how data is updated, the performance here is perfectly acceptable.

While trajectory bundling was greatly optimized for use in a real-time streamed data setting, limitations regarding implementation details left its performance short of our goals, with scalability being a problem as well. Given the performance, the recent period has to be kept at a short interval (30 minutes). However, this is still a lot of data, and it allows users to understand what is recent and what is not. However, the performance was well within our goals regarding the data processing for the History and Ongoing periods.

To improve this aspect of performance, further improvements would have to be done. These improvements would consist of further optimizing the algorithm and exploring a solution that does not require the use of a database, such as keeping data in memory and making a version of the PostGIS filter queries that was optimized for this type of data structure. While our use of geospatial database functions is what allowed the most significant increase in performance, it nevertheless holds us back somewhat.

5.4 Summary

In this chapter, we discussed and presented the evaluation we performed on our prototype. We started by evaluating the performance. We tested the visualization framerate, which had satisfactory results. Afterward, we compared our optimized trajectory bundling algorithm with the algorithm we started with, concluding that we had made significant performance improvements. Finally, we tested the streaming simulation's overall performance and its data processing, where we concluded that further improvements were required.

We then described the usability testing procedure. We invited participants to perform a set of tasks that evaluated different aspects of the visualization. Our results were very good, as the participants successfully understood the time period concept and the different representations. Finally, we discussed the results and reflected on certain improvements that could be made.

Chapter 6

Conclusion

The availability of devices that can record geospatial data and their constant connection to the Internet has resulted in large amounts of data being produced every second and enabled us to visualize it in real-time.

After studying current works that try to solve this problem, we concluded that no visualization handles large amounts of data and does so in real-time. We identified each solution's positive and negative aspects in order to develop a solution. We decided that to handle data as it is streamed in, we would have to divide this data into different periods of time and represent them in visually distinct ways.

We then studied different ways to represent and process each period. For one of these periods, we used trajectory bundling, an algorithm that bundles similar lines and represents them as a single line, leading to much improved visual clarity on a map, as well as less memory usage. For another period, we used grid binning, which creates a grid where each square has data pertaining to how many objects passed through it.

Afterward, we looked at how to represent the data produced by these algorithms. We ended up using three different time periods: Ongoing (trajectories happening currently), Recent (trajectories of objects that stopped recently), and History (trajectories of objects that stopped a longer time ago). For the Ongoing period, we showed the data as is, lines for each object's trajectory and points for its current location. As for the Recent period, we presented the lines produced by the trajectory bundling algorithm. Finally, we created two representations for the History period: a cluster heatmap that presents data in discrete squares and a spatial heatmap that creates continuous surfaces.

We created a prototype to implement these visualization and processing techniques. This prototype consists of a server that processes data and continually sends new data to the client visualization. The

server creates separate processes to take advantage of multi-core processors, and the client visualization uses a WebGL-based mapping library to maintain good performance when displaying large amounts of data.

After developing the prototype, we evaluated its usability by inviting users to perform a set of tasks and compare each History period representation. Tasks that required users to point out specific places in the History period were the hardest, with the spatial heatmap having better results than the cluster heatmap. Another complex task was observing how trends changed over time. Each user answered a questionnaire, including the raw NASA-TLX and SUS tests, which let us conclude that our prototype's usability was overall very good.

The representations for the ongoing and recent periods were both very easily understood, while the representations for the history period were slightly harder to understand.

As well as usability, we also evaluated performance. The visualization's performance (frame rate) was very stable, with slight stutters when loading data. While we optimized our trajectory bundling algorithm significantly when compared to Understand My Steps (Sil and Gonçalves, 2018), its scalability of the Recent period was below our objectives.

The performance in the recent period stemmed from structure limitations. We used database queries to decrease the time spent performing trajectory bundling significantly, but this improvement still came at the cost of having to interact with the database, which resulted in an overhead. However, the Ongoing and History periods proved to have very good performance.

While there were limitations regarding processing performance and scalability, these were only for one of the periods we developed. All other goals were met, meaning we succeeded at creating a visualization that shows large amounts of data as it evolves in real-time, and this visualization was easily understandable.

6.1 Future Work

Given the results of the performance and usability tests, we believe some changes could be performed.

The prototype structure could be changed so that trajectory bundled data was stored in memory and not the database, resulting in faster access times. This would require developing a data filtering technique that works on data stored in process memory, and one would have to ensure the performance is not inferior.

Additionally, data processing in the visualization could be further optimized to avoid stuttering when

new data is received. The way animations for the Ongoing period line updates are prepared could be further optimized as well.

Regarding usability, further improvements could be made to the representations in the history period. For the spatial heatmap, adding a hover tooltip to indicate how many objects a certain part of the heatmap represents could be done. For the cluster heatmap, the color scale could be changed to make it easier to differentiate different squares.

Moreover, we can implement a way to let the user switch between heatmaps so they can pick whichever they prefer. Since each heatmap is only a different representation of the same data, we can also experiment with showing different heatmaps at different zoom levels: when zoomed out, the spatial heatmap could be shown to display the overall trends, but as the user zooms in it could switch to the cluster heatmap to allow the analysis of each individual square.

Finally, the graceful degradation concept we discussed could be further explored. More granular forms of this degradation, allowing for more and more configurable periods, could be made. Our modular time period concept could be built upon to incorporate periods with different types of trajectory simplification, as well as different representations.

Bibliography

Andrienko, G., Andrienko, N., Bak, P., Keim, D., and Wrobel, S. (2013). *Visual Analytics of Movement*. Springer, Berlin.

Andrienko, G., Andrienko, N., Chen, W., Maciejewski, R., and Zhao, Y. (2017). Visual analytics of mobility and transportation: State of the art and further research directions. *IEEE Transactions on Intelligent Transportation Systems*.

Ankerst, M., Breunig, M., Kriegel, H., and Sander, J. (1999). OPTICS: Ordering points to identify the clustering structure. *SIGMOD International Conference on Management of Data*.

Bangor, A., Kortum, P., and Miller, J. (2009). Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123.

Bangor, A., Kortum, P. T., and Miller, J. T. (2008). An empirical evaluation of the system usability scale. *International Journal of Human–Computer Interaction*, 24(6):574–594.

Bonferroni, C. (1936). Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, 8:3–62.

Bustamante, E. A. and Spain, R. D. (2008). Measurement invariance of the nasa tlx. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 52(19):1522–1526.

Chen, S., Yuan, X., Wang, Z., Guo, C., Liang, J., Wang, Z., Zhang, X., and Zhang, J. (2016). Interactive visual discovering of movement patterns from sparsely sampled geo-tagged social media data. *IEEE Transactions on Visualization and Computer Graphics*.

Chen, W., Huang, Z., Wu, F., Zhu, M., Guan, H., and Maciejewski, R. (2018). VAUD: A visual analysis approach for exploring spatio-temporal urban data. *IEEE Transactions on Visualization and Computer Graphics*.

Cruz, J. and Gonçalves, D. (2014). FlexTrans: Solução de suporte a serviços flexíveis de transporte de passageiros. Master's thesis, Instituto Superior Técnico.

Deng, Z., Hu, Y., Zhu, M., Huang, X., and Du, B. (2015). A scalable and fast optics for clustering trajectory big data. *Cluster Computing*.

Ester, M., peter Kriegel, H., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press.

Fialho, G. and Gonçalves, D. (2018). Visbig: Visualizar bigdata em tempo real. Master's thesis, Instituto Superior Técnico.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701.

Gil, R. and Goncalves, D. (2016). GatherMySteps. In *2016 23rd Portuguese Meeting on Computer Graphics and Interaction*.

Gomes, G. A. M., Santos, E., and Vidal, C. A. (2017). Interactive visualization of traffic dynamics based on trajectory data. In *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images*.

Guo, H., Wang, Z., Yu, B., Zhao, H., and Yuan, X. (2011). TripVista: Triple perspective visual trajectory analytics and its application on microscopic traffic data at a road intersection. In *2011 IEEE Pacific Visualization Symposium*.

Harrower, M. and Brewer, C. A. (2003). Colorbrewer.org: An online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37.

Havre, S., Hetzler, E., Whitney, P., and Nowell, L. (2002). ThemeRiver: visualizing thematic changes in large document collections. *IEEE Transactions on Visualization and Computer Graphics*.

He, J., Chen, H., Chen, Y., Tang, X., and Zou, Y. (2019). A review of variable-based spatiotemporal trajectory data visualization. *IEEE Access*.

Huang, X., Zhao, Y., Ma, C., Yang, J., Ye, X., and Zhang, C. (2016). TrajGraph: A graph-based visual analytics approach to studying urban network centralities using taxi trajectory data. *IEEE Transactions on Visualization and Computer Graphics*.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of psychology*.

Lou, Y., Zhang, C., Zheng, Y., Xie, X., Wang, W., and Huang, Y. (2009). Map-matching for low-sampling-rate gps trajectories. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09.

Mao, J., Song, Q., Jin, C., Zhang, Z., and Zhou, A. (2018). Online clustering of streaming trajectories. *Frontiers of Computer Science*.

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., and Damas, L. (2013). Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402.

Munzner, T. (2014). *Visualization Analysis and Design*. AK Peters Visualization Series.

Patwary, M. A., Palsetia, D., Agrawal, A., Liao, W., Manne, F., and Choudhary, A. (2013). Scalable parallel optics data clustering using graph algorithmic techniques. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13.

Rinzivillo, S., Pedreschi, D., Nanni, M., Giannotti, F., Andrienko, N., and Andrienko, G. (2008). Visually driven analysis of movement data by progressive clustering. *Information Visualization*.

Saalfeld, A. (1999). Topologically consistent line simplification with the douglas-peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18.

Sacharidis, D., Patroumpas, K., Terrovitis, M., Kantere, V., Potamias, M., Mouratidis, K., and Sellis, T. (2008). On-line discovery of hot motion paths. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '08.

Scheepens, R., Willems, N., van de Wetering, H., and van Wijk, J. (2012). Interactive density maps for moving objects. *IEEE Computer Graphics and Applications*.

Shapiro, S. S. and Wilk, M. B. (1965). An analysis of variance test for normality (complete samples)†. *Biometrika*, 52(3-4):591–611.

Sil, D. and Gonçalves, D. (2018). Understand my steps: Using edge-bundling to visualize GPS tracks. Master's thesis, Instituto Superior Técnico.

Steptoe, M., Krüger, R., Garcia, R., Liang, X., and Maciejewski, R. (2018). A visual analytics framework for exploring theme park dynamics. *Transactions on Interactive Intelligent Systems*.

Wang, Z., Lu, M., Yuan, X., Zhang, J., and v. d. Wetering, H. (2013). Visual traffic jam analysis based on trajectory data. *IEEE Transactions on Visualization and Computer Graphics*.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.

Wilkinson, L. and Friendly, M. (2009). The history of the cluster heat map. *The American Statistician*, 63(2):179–184.

Yang, C., Zhang, Y., Tang, B., and Zhu, M. (2019). Vaite: A visualization-assisted interactive big urban trajectory data exploration system. In *2019 IEEE 35th International Conference on Data Engineering*.

Zhao, Y., Al-Dohuki, S., Eynon, T., Kamw, F., Sheets, D., Ma, C., Hu, Y., Ye, X., and Yang, J. (2016). TrajAnalytics: A web-based visual analytics software of urban trajectory data. In *2016 IEEE Visualization Conference*.

Appendix A

User questionnaire

Below is the form users filled in the user testing process. Since all users were Portuguese, the form is in Portuguese as well.

FastGeo - Questionário de avaliação de utilizador

Está a ser convidado para participar no estudo de um protótipo de visualização no âmbito de uma dissertação de Mestrado em Engenharia Informática e de Computadores.

O objetivo deste estudo é compreender como mostrar grandes quantidades de dados geoespaciais numa maneira clara e fácil de compreender, permitindo que utilizadores identifiquem tendências nestes dados, e que percebam como estas tendências mudam com a passagem do tempo.

Ao aceitar a participação neste estudo, ser-lhe-á pedido que participe num conjunto de tarefas. Nenhuma destas tarefas requer experiência prévia ou conhecimentos específicos desta área. A sua interação com o protótipo será gravada para análise posterior.

Quaisquer dados recolhidos durante este processo serão mantidos em privado, e apenas os responsáveis pelo estudo terão acesso a eles. Caso alguma informação seja publicada, a sua identificação não será possível.

A participação neste estudo é voluntária. Se decidir participar no mesmo, é livre de desistir a qualquer momento, ou de saltar qualquer tarefa que não queira fazer.

* Required

Caracterização

1. ID *

2. Qual é a sua idade? *

3. Qual é o seu género? *

   Mark only one oval.

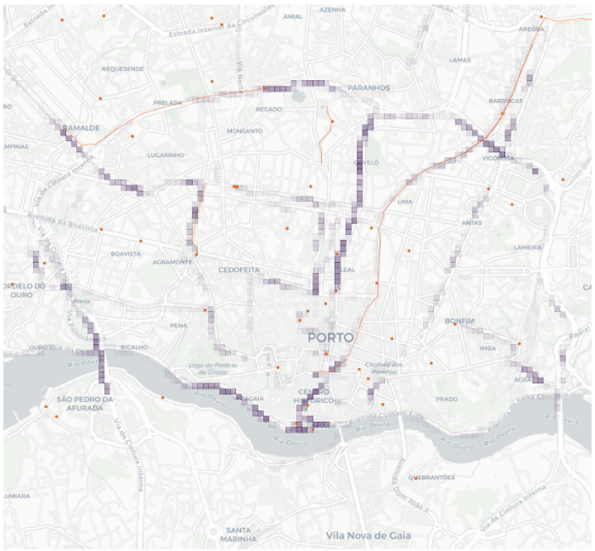
   - Feminino
   - Masculino
   - Prefiro não dizer

4. Dadas as seguintes imagens, por favor indique o número que vê nas mesmas. Se não conseguir ver um número, responda com um traço (-). Separe cada número com uma vírgula (ex: 59, 20, -). *

Por favor complete as tarefas que lhe forem pedidas antes de prosseguir para a próxima parte do questionário.

Funcionalidades FastGeo

Representações do período Histórico



A

B

5. Achei o período Em Curso fácil de perceber. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

6. Achei o período Recente fácil de perceber. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

7. Achei a representação A do período Histórico fácil de perceber. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

8. Achei a representação B do período Histórico fácil de perceber. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

9. Percebi bem a transição do período Em Curso para o período Recente. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

10. Percebi bem a transição do período Recente para o período Histórico (com a representação A). *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

11. Percebi bem a transição do período Recente para o período Histórico (com a representação B). *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

12. Achei fácil diferenciar os três diferentes períodos (com a representação A do período Histórico). *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

13. Achei fácil diferenciar os três diferentes períodos (com a representação B do período Histórico). *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

14. Tem alguma sugestão para novas técnicas de representação ou funcionalidades?

NASA-TLX

Nesta secção, por favor responda um número de 1 a 20 a todas as perguntas.

15. Exigência mental: qual foi a exigência mental (ex: pensar, decidir, calcular, recordar, olhar, procurar, etc?) *
(1 - baixa; 20 - alta)

16. Exigência física: qual foi a exigência física (ex: empurrar, puxar, virar, controlar, ativar, etc)? As tarefas foram cansativas fisicamente? *
(1 - baixa; 20 - alta)

17. Exigência temporal: quanta pressão de tempo sentiu devido ao ritmo a que as tarefas foram executadas? O ritmo foi calmo ou acelerado? *
(1 - baixa; 20 - alta)

18. Desempenho: como classifica o sucesso na resolução das tarefas? Quão satisfeito está com a sua resolução das tarefas? *
(1 - bom; 20 - fraco)

_____

19. Esforço: qual foi o nível de esforço (físico e mental) que necessitou para atingir o seu desempenho? *
(1 - baixo; 20 - alto)

_____

20. Frustração: quão inseguro, desencorajado, irritado, stressado e chateado se sentiu, em relação a quão seguro, gratificado, contente, relexado e calmo se sentiu durante as suas tarefas? *
(1 - baixa; 20 - alta)

_____

System Usability Scale

21. Acho que gostaria de utilizar este sistema com frequência. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

22. Considerei o sistema mais complexo do que necessário. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

23. Achei o sistema fácil de utilizar. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

24. Acho que necessitaria de ajuda de um técnico para conseguir utilizar este sistema. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

25. Considerei que as várias funcionalidades deste sistema estavam bem integradas. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

26. Achei que este sistema tinha muitas inconsistências. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente	◯	◯	◯	◯	◯	Concordo fortemente

27. Suponho que a maioria das pessoas aprenderia a utilizar rapidamente este sistema. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente						Concordo fortemente

28. Considerei o sistema muito complicado de utilizar. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente						Concordo fortemente

29. Senti-me muito confiante a utilizar este sistema. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente						Concordo fortemente

30. Tive que aprender muito antes de conseguir lidar com este sistema. *

Mark only one oval.

	1	2	3	4	5	
Discordo fortemente						Concordo fortemente

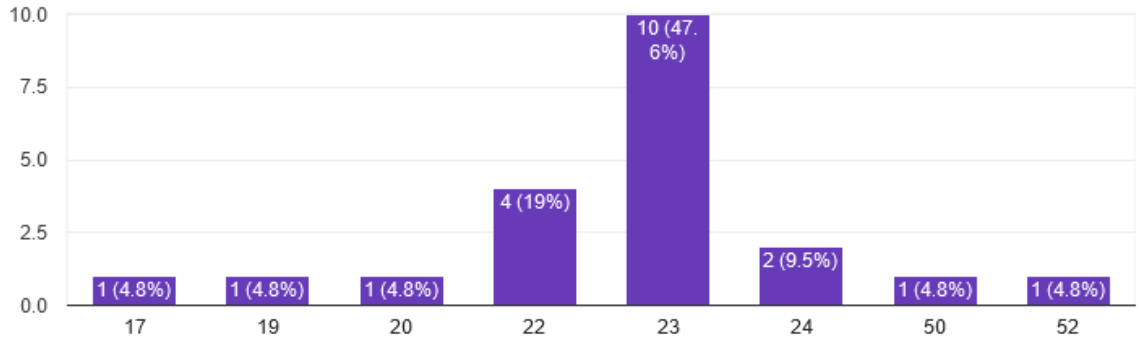
Appendix B

Questionnaire Results

Below are the results for the user questionnaire.

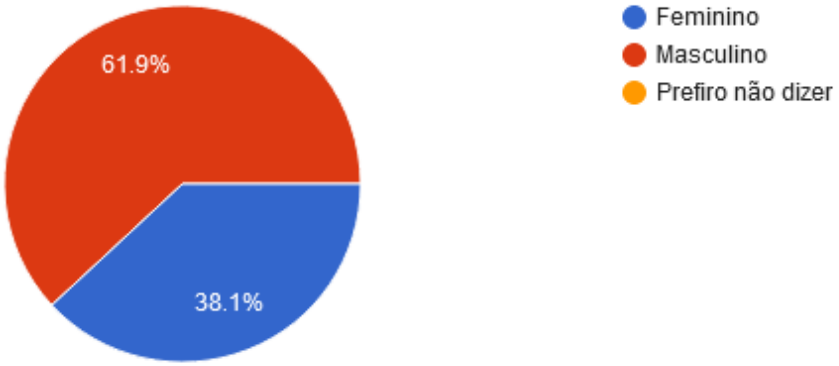
Qual é a sua idade?

21 responses



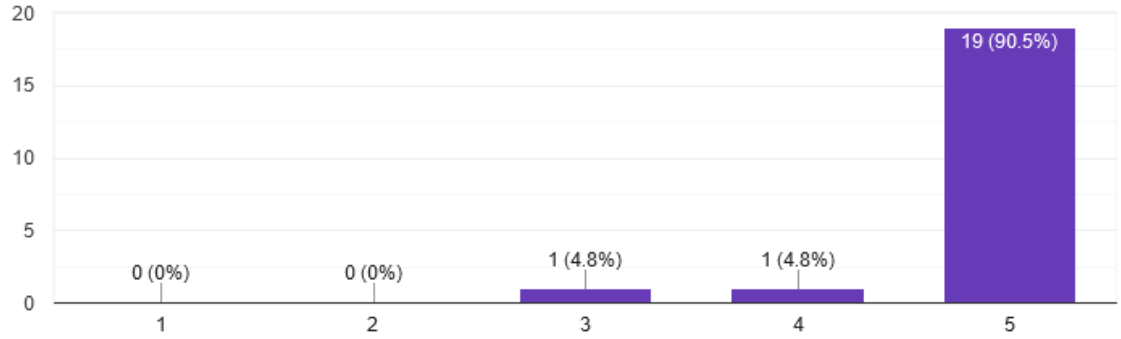
Qual é o seu género?

21 responses



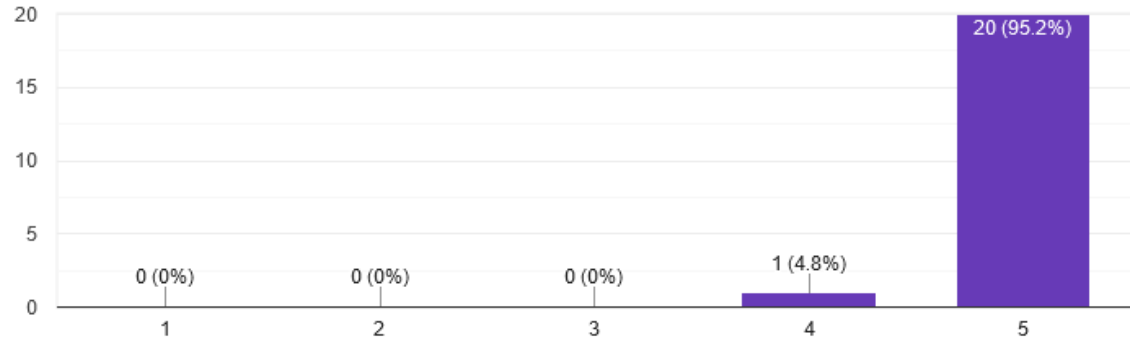
Achei o período Em Curso fácil de perceber.

21 responses



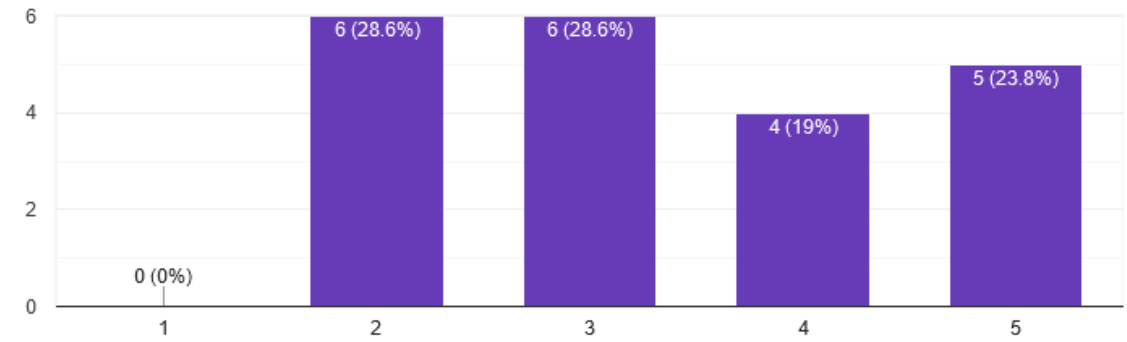
Achei o período Recente fácil de perceber.

21 responses



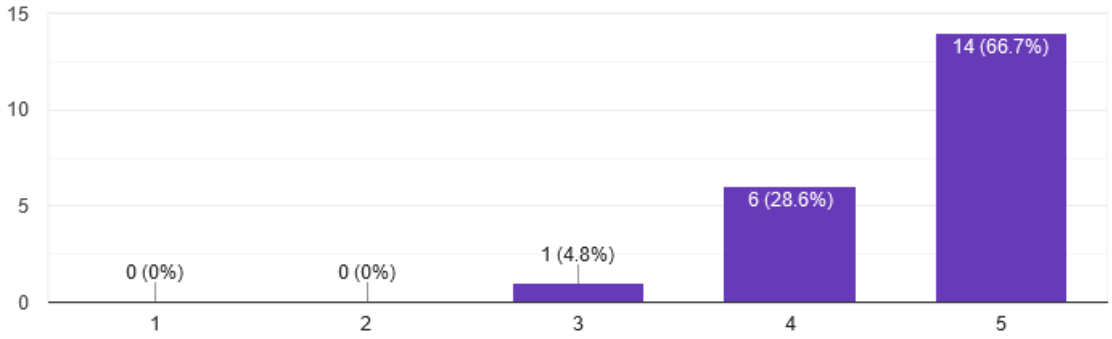
Achei a representação A do período Histórico fácil de perceber.

21 responses



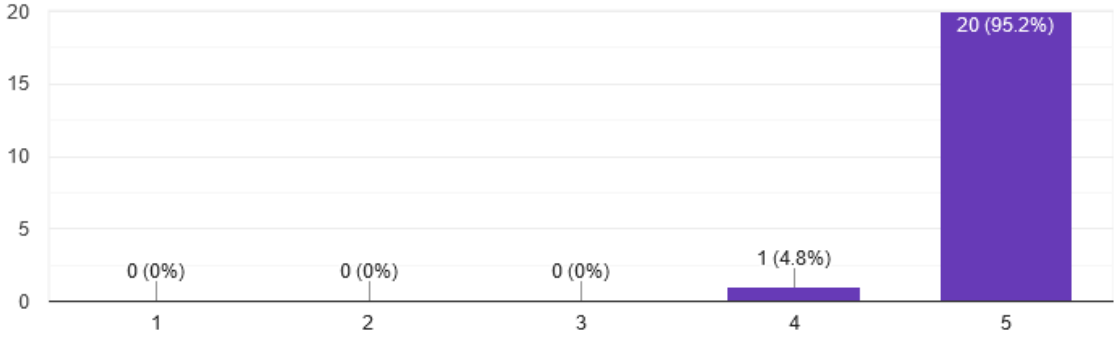
Achei a representação B do período Histórico fácil de perceber.

21 responses



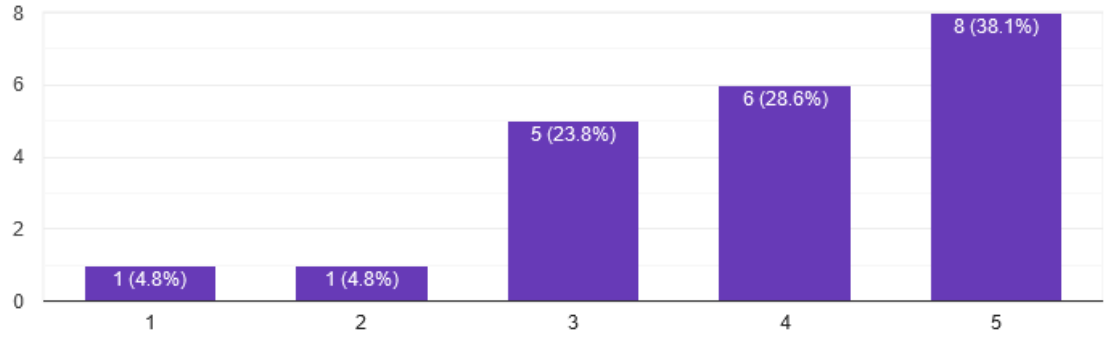
Percebi bem a transição do período Em Curso para o período Recente.

21 responses



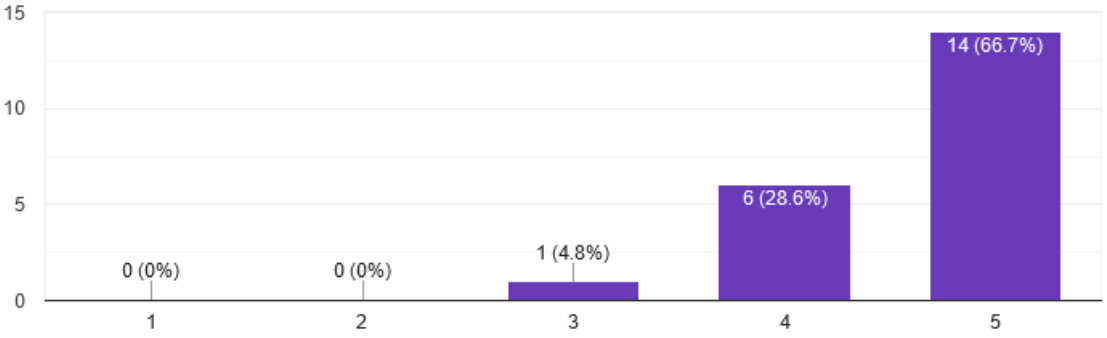
Percebi bem a transição do período Recente para o período Histórico (com a representação A).

21 responses



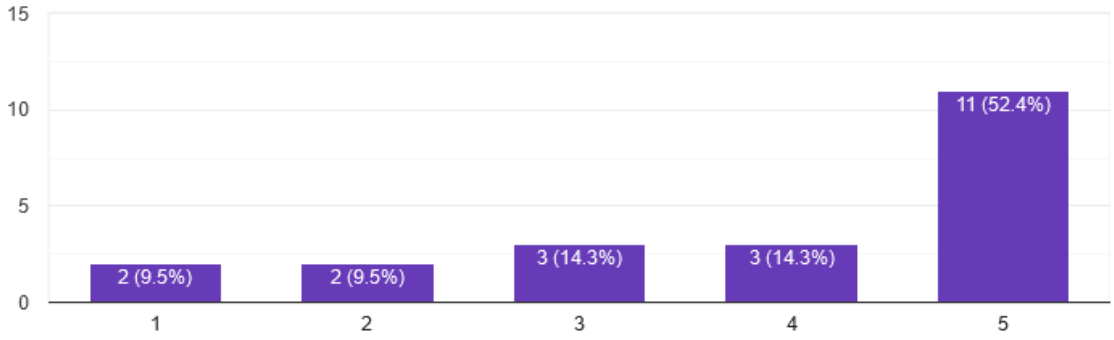
Percebi bem a transição do período Recente para o período Histórico (com a representação B).

21 responses



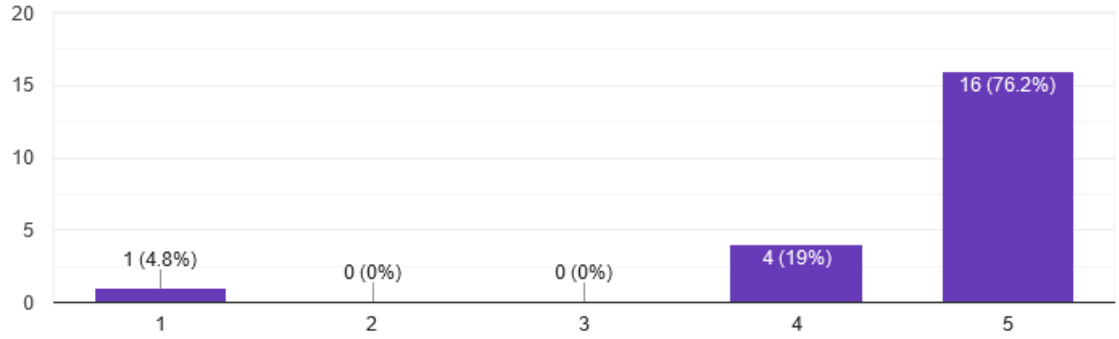
Achei fácil diferenciar os três diferentes períodos (com a representação A do período Histórico).

21 responses



Achei fácil diferenciar os três diferentes períodos (com a representação B do período Histórico).

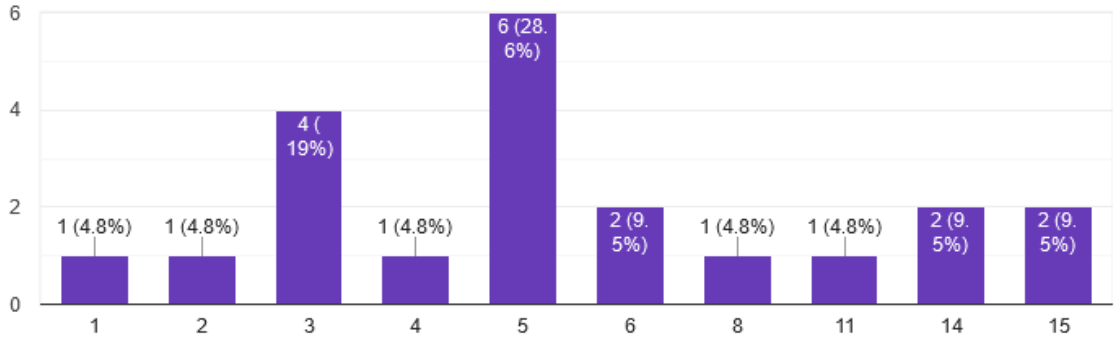
21 responses



NASA-TLX

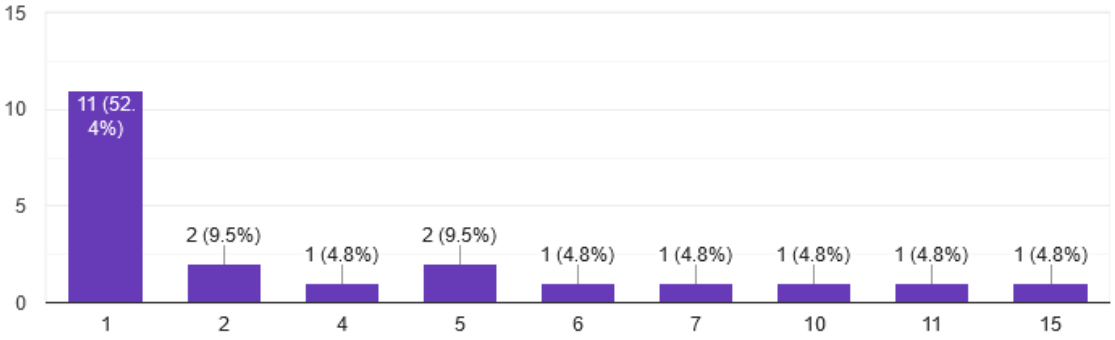
Exigência mental: qual foi a exigência mental (ex: pensar, decidir, calcular, recordar, olhar, procurar, etc?)

21 responses



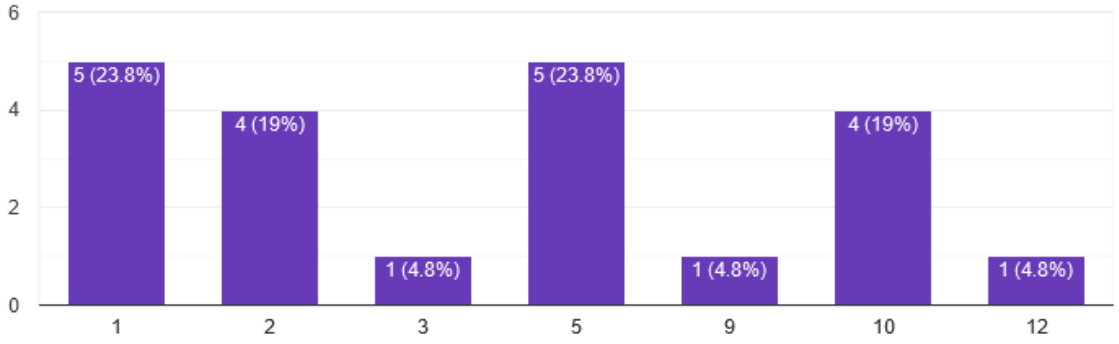
Exigência física: qual foi a exigência física (ex: empurrar, puxar, virar, controlar, ativar, etc)? As tarefas foram cansativas fisicamente?

21 responses



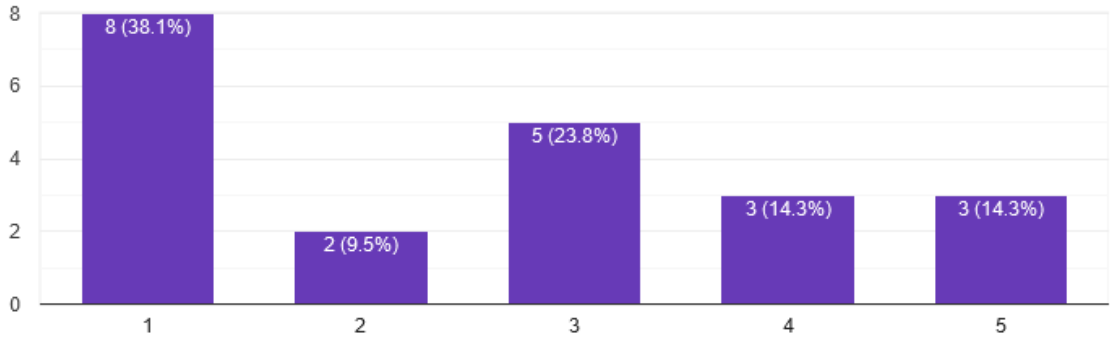
Exigência temporal: quanta pressão de tempo sentiu devido ao ritmo a que as tarefas foram executadas? O ritmo foi calmo ou acelerado?

21 responses



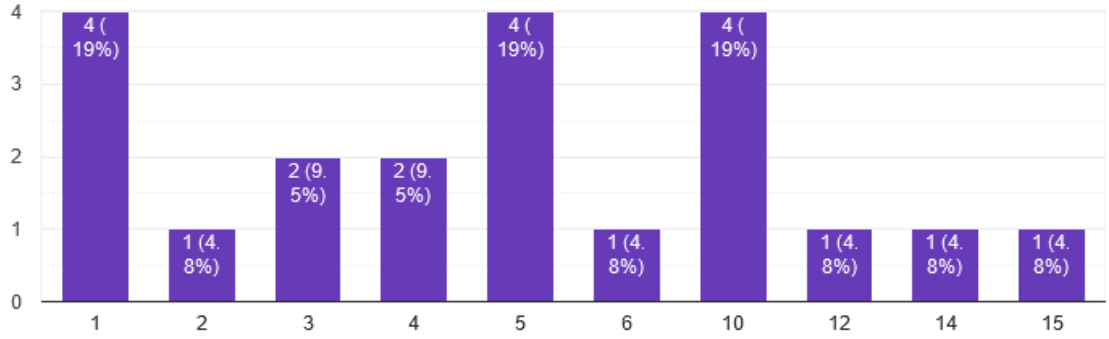
Desempenho: como classifica o sucesso na resolução das tarefas? Quão satisfeito está com a sua resolução das tarefas?

21 responses



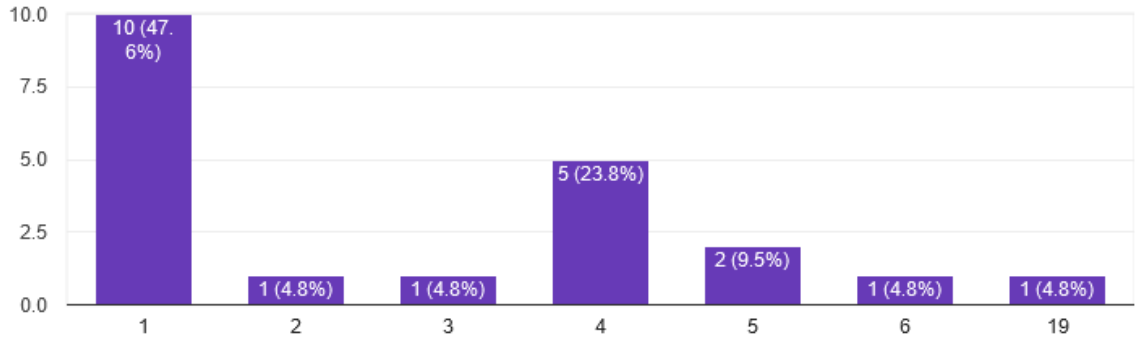
Esforço: qual foi o nível de esforço (físico e mental) que necessitou para atingir o seu desempenho?

21 responses



Frustração: quão inseguro, desencorajado, irritado, stressado e chateado se sentiu, em relação a quão seguro, gratificado, contente, relexado e calmo se sentiu durante as suas tarefas?

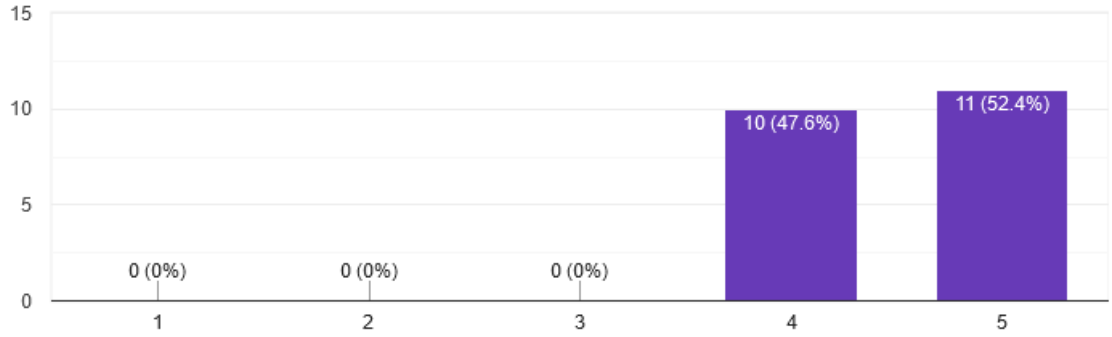
21 responses



System Usability Scale

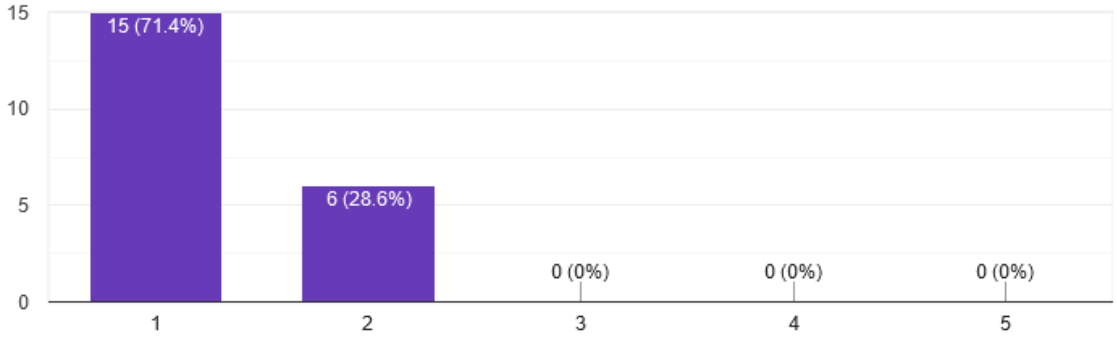
Acho que gostaria de utilizar este sistema com frequência.

21 responses



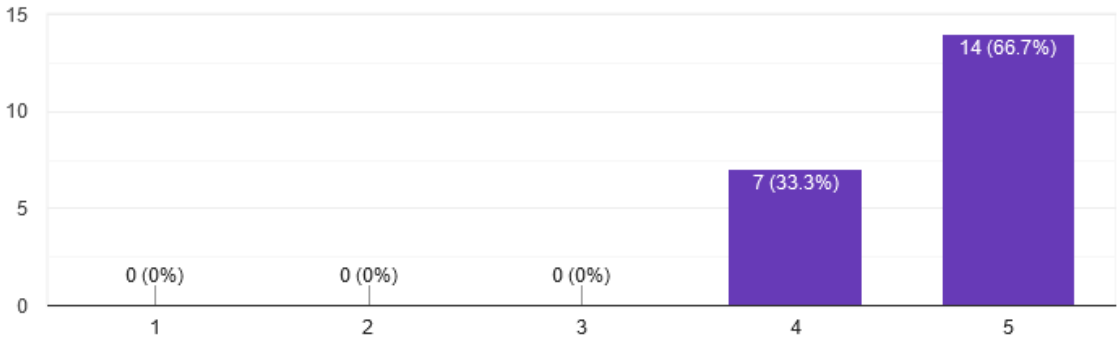
Considerei o sistema mais complexo do que necessário.

21 responses



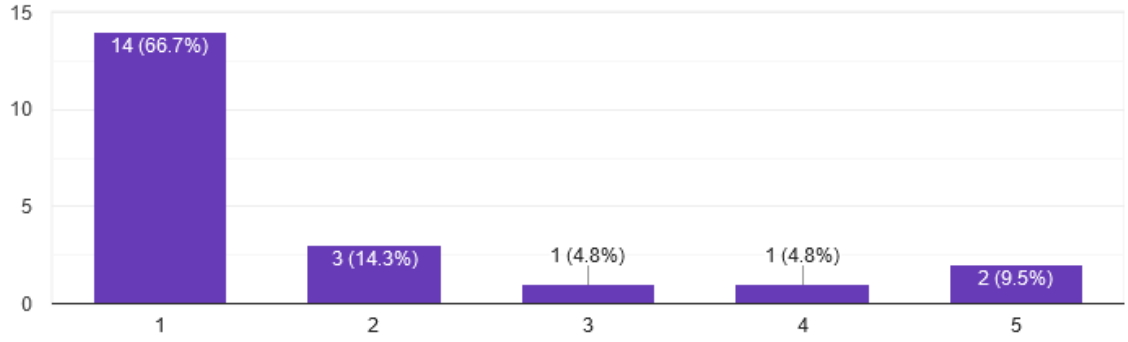
Achei o sistema fácil de utilizar.

21 responses



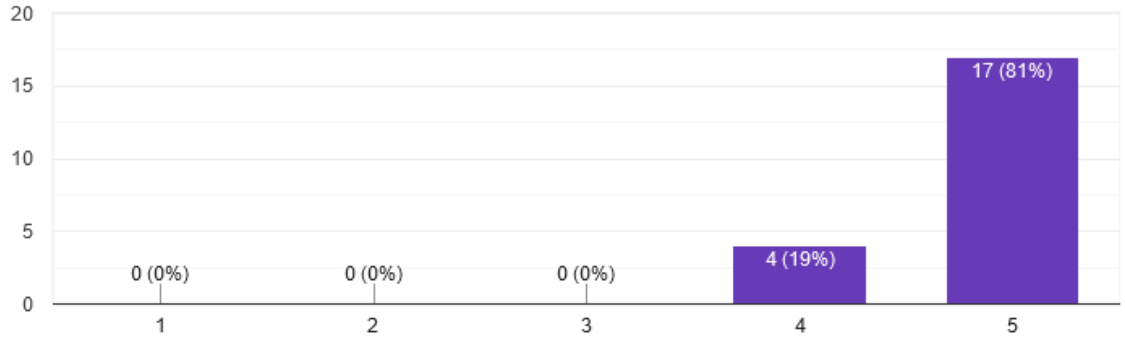
Acho que necessitaria de ajuda de um técnico para conseguir utilizar este sistema.

21 responses



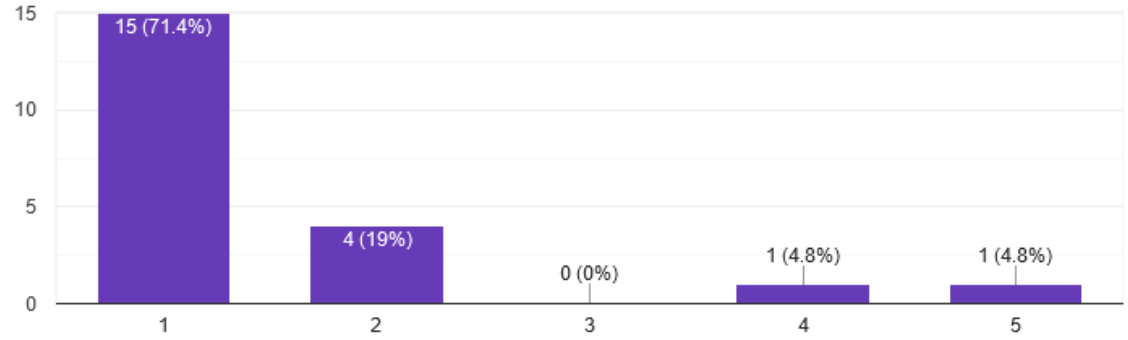
Considerei que as várias funcionalidades deste sistema estavam bem integradas.

21 responses



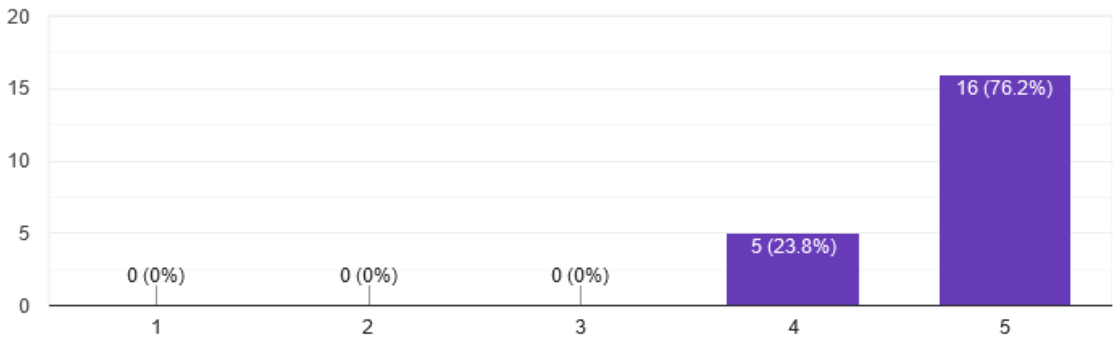
Achei que este sistema tinha muitas inconsistências.

21 responses



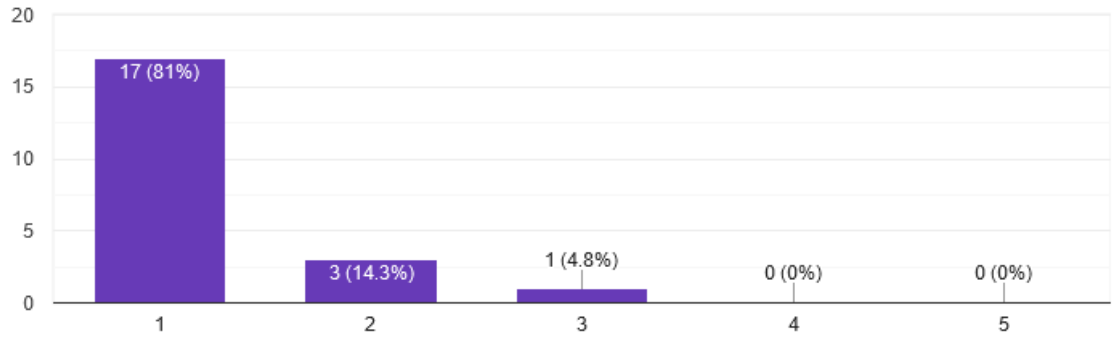
Suponho que a maioria das pessoas aprenderia a utilizar rapidamente este sistema.

21 responses



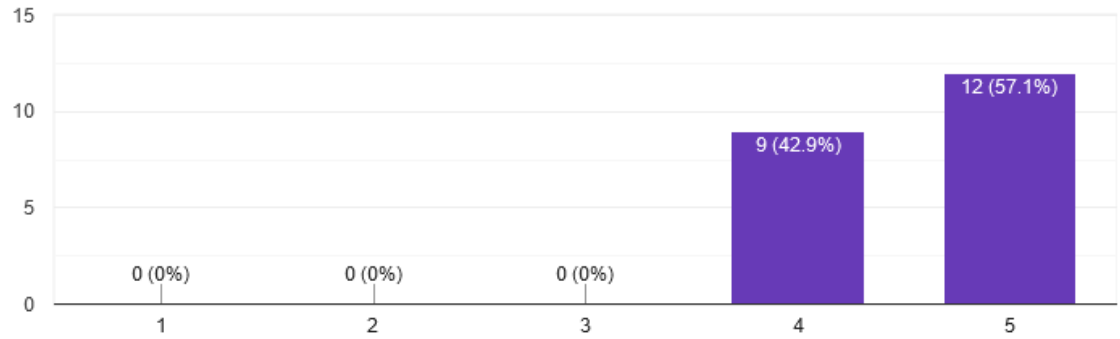
Considerei o sistema muito complicado de utilizar.

21 responses



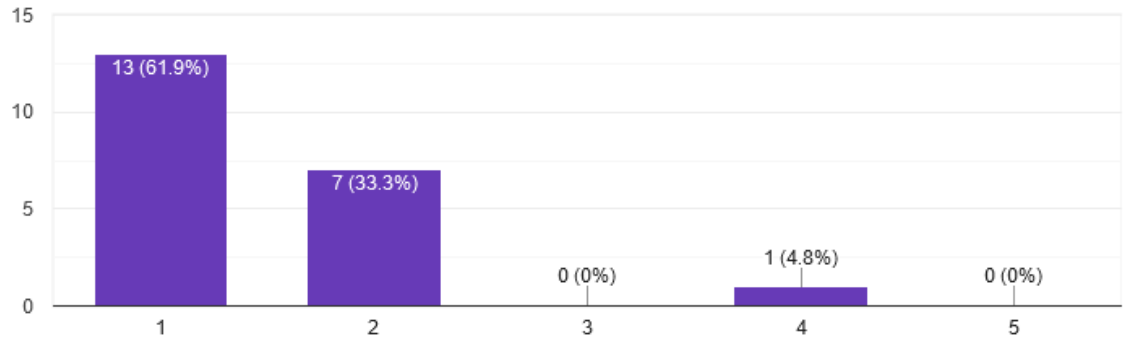
Senti-me muito confiante a utilizar este sistema.

21 responses



Tive que aprender muito antes de conseguir lidar com este sistema.

21 responses



Appendix C

Statistical Analysis

Below are the complete results of the statistical analysis. Tasks 30 and 60 are tasks 3 and 6 with the spatial heatmap representation, while Tasks 31 and 61 are Tasks 3 and 6 with the cluster heatmap representation.

Ranks

		N	Mean Rank	Sum of Ranks
T31 - T30	Negative Ranks	8[a]	10,13	81,00
	Positive Ranks	13[b]	11,54	150,00
	Ties	0[c]		
	Total	21		
D31 - D30	Negative Ranks	12[d]	7,75	93,00
	Positive Ranks	3[e]	9,00	27,00
	Ties	6[f]		
	Total	21		
T61 - T60	Negative Ranks	9[j]	9,17	82,50
	Positive Ranks	12[k]	12,38	148,50
	Ties	0[l]		
	Total	21		
D61 - D60	Negative Ranks	6[p]	4,08	24,50
	Positive Ranks	1[q]	3,50	3,50
	Ties	14[r]		
	Total	21		

	T31 - T30	D31 - D30	T61 - T60	D61 - D60
Z	-1,199[b]	-1,995[c]	-1,148[b]	-1,897[c]
Asymp. Sig. (2-tailed)	,230	,046	,251	,058

Figure C.1: Pairwise comparisons of time (T) and difficulty (D) between both versions of task 3 and 6.

S30 & S31

S30	S31 0	S31 1
0	1	2
1	5	13

Test Statistics[a]

	S30 & S31
N	21
Exact Sig. (2-tailed)	,453[b]

a. McNemar Test

b. Binomial distribution used.

Figure C.2: Pairwise comparison of success between both versions of task 3.

Ranks

	Mean Rank
T1	1,00
T2	4,76
T30	8,76
T31	9,71
T4	3,88
T5	5,07
T60	6,50
T61	6,90
T7	5,00
T8	4,00
T9	10,76
T10	11,64

Test Statistics[a]

N	21
Chi-Square	179,588
df	11
Asymp. Sig.	,000

a. Friedman Test

Figure C.3: Results of Friedman test for every task's time.

Test Statistics[a]

	T2 - T1	T30 - T1	T31 - T1	T4 - T1	T5 - T1	T60 - T1
Z	-4,017[b]	-4,015[b]	-4,015[b]	-4,019[b]	-4,017[b]	-4,016[b]
Asymp. Sig. (2-tailed)	,000	,000	,000	,000	,000	,000

Test Statistics[a]

	T61 - T1	T7 - T1	T8 - T1	T9 - T1	T10 - T1	T30 - T2
Z	-4,016[b]	-4,016[b]	-4,021[b]	-4,016[b]	-4,015[b]	-3,946[b]
Asymp. Sig. (2-tailed)	,000	,000	,000	,000	,000	,000

Test Statistics[a]

	T31 - T2	T4 - T2	T5 - T2	T60 - T2	T61 - T2	T7 - T2
Z	-4,015[b]	-,817[c]	-,070[b]	-2,195[b]	-2,627[b]	-,302[b]
Asymp. Sig. (2-tailed)	,000	,414	,945	,028	,009	,762

Test Statistics[a]

	T8 - T2	T9 - T2	T10 - T2	T4 - T30	T5 - T30	T60 - T30
Z	-1,305[c]	-4,015[b]	-4,015[b]	-3,981[c]	-3,659[c]	-3,494[c]
Asymp. Sig. (2-tailed)	,192	,000	,000	,000	,000	,000

Test Statistics[a]

	T61 - T30	T7 - T30	T8 - T30	T9 - T30	T10 - T30	T4 - T31
Z	-2,416[c]	-3,772[c]	-3,883[c]	-2,677[b]	-3,841[b]	-4,015[c]
Asymp. Sig. (2-tailed)	,016	,000	,000	,007	,000	,000

Test Statistics[a]

	T5 - T31	T60 - T31	T61 - T31	T7 - T31	T8 - T31	T9 - T31
Z	-4,015[c]	-3,980[c]	-3,454[c]	-4,015[c]	-4,015[c]	-2,590[b]
Asymp. Sig. (2-tailed)	,000	,000	,001	,000	,000	,010

Test Statistics[a]

	T10 - T31	T5 - T4	T60 - T4	T61 - T4	T7 - T4	T8 - T4
Z	-3,696[b]	-1,547[b]	-2,353[b]	-2,988[b]	-1,289[b]	,000[d]
Asymp. Sig. (2-tailed)	,000	,122	,019	,003	,197	1,000

Test Statistics[a]

	T9 - T4	T10 - T4	T60 - T5	T61 - T5	T7 - T5	T8 - T5
Z	-4,015[b]	-4,015[b]	-2,278[b]	-2,990[b]	-,149[b]	-1,607[c]
Asymp. Sig. (2-tailed)	,000	,000	,023	,003	,881	,108

Test Statistics[a]

	T9 - T5	T10 - T5	T61 - T60	T7 - T60	T8 - T60	T9 - T60
Z	-4,015[b]	-4,015[b]	-1,148[b]	-2,943[c]	-2,819[c]	-4,015[b]
Asymp. Sig. (2-tailed)	,000	,000	,251	,003	,005	,000

Test Statistics[a]

	T10 - T60	T7 - T61	T8 - T61	T9 - T61	T10 - T61	T8 - T7
Z	-4,015[b]	-2,922[c]	-3,112[c]	-4,015[b]	-4,015[b]	-1,601[c]
Asymp. Sig. (2-tailed)	,000	,003	,002	,000	,000	,109

Test Statistics[a]

	T9 - T7	T10 - T7	T9 - T8	T10 - T8	T10 - T9
Z	-4,015[b]	-4,015[b]	-4,015[b]	-4,015[b]	-3,285[b]
Asymp. Sig. (2-tailed)	,000	,000	,000	,000	,001

a. Wilcoxon Signed Ranks Test

b. Based on negative ranks.

c. Based on positive ranks.

d. The sum of negative ranks equals the sum of positive ranks.

Figure C.4: Pairwise comparisons of time between every task.

Ranks

	Mean Rank
D1	7,93
D2	7,45
D30	4,10
D31	3,00
D4	7,00
D5	6,69
D60	6,69
D61	5,45
D7	7,21
D8	7,02
D9	3,45

Test Statistics[a]

N	21
Chi-Square	86,923
df	10
Asymp. Sig.	,000

a. Friedman Test

Figure C.5: Results of Friedman test for every task's difficulty.

Test Statistics[a]

	D2 - D1	D30 - D1	D31 - D1	D4 - D1	D5 - D1	D60 - D1
Z	-1,414[b]	-3,900[b]	-3,834[b]	-1,633[b]	-2,121[b]	-2,070[b]
Asymp. Sig. (2-tailed)	,157	,000	,000	,102	,034	,038

Test Statistics[a]

	D61 - D1	D7 - D1	D8 - D1	D9 - D1	D30 - D2	D31 - D2
Z	-2,919[b]	-1,633[b]	-2,000[b]	-3,487[b]	-3,638[b]	-3,624[b]
Asymp. Sig. (2-tailed)	,004	,102	,046	,000	,000	,000

Test Statistics[a]

	D4 - D2	D5 - D2	D60 - D2	D61 - D2	D7 - D2	D8 - D2
Z	-,966[b]	-1,414[b]	-1,406[b]	-2,495[b]	-,816[b]	-1,000[b]
Asymp. Sig. (2-tailed)	,334	,157	,160	,013	,414	,317

Test Statistics[a]

	D9 - D2	D31 - D30	D4 - D30	D5 - D30	D60 - D30	D61 - D30
Z	-3,508[b]	-1,995[b]	-2,828[c]	-2,236[c]	-2,357[c]	-,894[c]
Asymp. Sig. (2-tailed)	,000	,046	,005	,025	,018	,371

Test Statistics[a]

	D7 - D30	D8 - D30	D9 - D30	D4 - D31	D5 - D31	D60 - D31
Z	-3,153[c]	-3,606[c]	-1,698[b]	-3,030[c]	-3,201[c]	-3,201[c]
Asymp. Sig. (2-tailed)	,002	,000	,090	,002	,001	,001

Test Statistics[a]

	D61 - D31	D7 - D31	D8 - D31	D9 - D31	D5 - D4	D60 - D4
Z	-2,841[c]	-3,624[c]	-3,448[c]	-,626[c]	-,333[b]	-,707[b]
Asymp. Sig. (2-tailed)	,005	,000	,001	,531	,739	,480

Test Statistics[a]

	D61 - D4	D7 - D4	D8 - D4	D9 - D4	D60 - D5	D61 - D5
Z	-1,999[b]	-,322[c]	-,333[c]	-3,094[b]	,000[d]	-1,897[b]
Asymp. Sig. (2-tailed)	,046	,748	,739	,002	1,000	,058

Test Statistics[a]

	D7 - D5	D8 - D5	D9 - D5	D61 - D60	D7 - D60	D8 - D60
Z	-1,342[c]	-,632[c]	-3,153[b]	-1,897[b]	-1,000[c]	-,966[c]
Asymp. Sig. (2-tailed)	,180	,527	,002	,058	,317	,334

Test Statistics[a]

	D9 - D60	D7 - D61	D8 - D61	D9 - D61	D8 - D7	D9 - D7
Z	-3,090[b]	-2,310[c]	-2,070[c]	-2,296[b]	,000[d]	-3,346[b]
Asymp. Sig. (2-tailed)	,002	,021	,038	,022	1,000	,001

Test Statistics[a]

	D9 - D8
Z	-3,407[b]
Asymp. Sig. (2-tailed)	,001

a. Wilcoxon Signed Ranks Test

b. Based on positive ranks.

c. Based on negative ranks.

d. The sum of negative ranks equals the sum of positive ranks.

Figure C.6: Pairwise comparisons of difficulty between every task.

Appendix D

Prototype Guide

Below is a short guide that explains how run the prototype, as well as detailing the folder structure and the time period module creation process. This guide is available on the prototype's public repository [1][2].

[1] README.md: `https://github.com/lucasrafael98/FastGeo/blob/main/README.md`, last visited December 30th, 2020
[2] GUIDE.md: `https://github.com/lucasrafael98/FastGeo/blob/main/docs/GUIDE.md`, last visited December 30th, 2020

FastGeo

Real-time visualization of large amounts of geospatial data.

Developed for an MSc dissertation supervised by professors Daniel Gonçalves and Daniel Mendes.

How to run

FastGeo requires Python 3, NodeJS and PostgreSQL (with PostGIS extensions).

Install any dependencies:

```
npm install
pip install pyyaml psycopg2 ujson
```

Place your data in `src/data/raw`.

In `index.js`, update your database credentials, and configure any settings you may wish to in `config.yaml`.

Run the Node.js server:

```
cd src/fastgeo
npm start
```

Open a browser and go to http://localhost:3000/.

For more details, check the guide.

FastGeo Guide

This guide covers the project structure of FastGeo. If any details elude you, please read the dissertation.

Structure

FastGeo is an Express project, so the structure is similar, except for the `preprocess` and `data` folders:

```
fastgeo
    |- bin
    |- data (doesn't show up on the repository)
    |- preprocess
    |- public
    |   |- javascripts
    |   |- stylesheets
    |   |- vendor
    |- routes
    |   |- _aux
    |   |- stream
    |   |   |- modules
    |- views
```

The `views` folder contains the front end views, which use JavaScript and CSS code from the `public` folder.

The `routes` folder contains back end JavaScript code. `index.js` handles front end requests, and starts the streaming simulation loop, which is implemented in `stream/manager.js`. Time period modules inherit the base code from `stream/modules/stream_module.js`. The `_aux` folder contains code for child process creation.

The `preprocess` folder contains all Python code. Files named `module_*.py` correspond to the Python side of the time period modules, and they communicate with JavaScript and call preprocessing methods found in the other files.

The `data` folder contains, obviously, data. The simulation takes as starting input whatever is on the `raw` folder. If you want to use a different dataset than the one used for this thesis, you may need to implement custom parsing for it in `prepreocessing/initial_parsing.py` so that the dataset is converted properly into something the simulation can use, though you won't have to change anything else.

Alternatively, you can set up a `resumeFolder` in `config.yaml`. These resume folders are created by simply stopping the simulation, which creates a `last` folder which you can then use to resume the simulation from the point at which you stopped.

Each simulation step, the manager (`routes/stream/manager.js`) will grab the data fetched by the `rawFetchLoop()`, and perform the three sub-steps by asynchronously calling each module's `removeData()` function, creating three promises, all of which the manager will wait for before then doing the same for `updateData()`, and then for `displayData()`.

After the `displayData()` methods end, the manager will join all the data that these return into `data/temp/update.json`, which is sent to the front end when requested. If the front end requests the same file twice (which happens if a simulation step has taken longer than the interval for front end requests), it will ignore the second/third/etc. time this file is sent.

Creating a new time period

To create a new time period, do the following:

- Create a new class in `routes/stream/modules` inheriting from the base interface
- Implement removal/update/display methods (more on this below)
- Create a `module_<modulename>.py` file to handle data processing
- Make sure you are instancing a class of your new module and calling its methods in `manager.js`
- Implement the Mapbox layer/source for the data it sends through `update.json` in the front end.

The removal/update/display methods in the JavaScript module code all follow the same basic structure. You'll want to return a Promise (so that the simulation manager can wait for the method to be done), and then you'll do the following before resolving the promise:

```
this.childProcess.stdin.write(`['<command>', <arg1>,
  <arg2>, ...]\n`, "utf-8");
let promise = new Promise((resolve,
  reject)=>{this.continue = resolve;})
await
  promise;
```

For each Python module, the basic code is:

```
while(keep_running):
    next_command =
  eval(input())
    if(next_command[0] ==
  "<command1>"):

  do_something(next_command[1:])
    elif(next_command[0] ==
  "<command2>"):

  do_something_else(next_command[1:])
```

```
    (...)

  print('_')
```

In each module, `this.childProcess` is its corresponding Python process. Communication is done by simple `stdin/stdout`. The JavaScript module writes a command in the Python process's `stdin`, which accepts the command with `input()` and does different things depending on it. After performing the command, it `print()`s an underscore `'_'`, which the JavaScript module takes as a signal that it can resolve its promise and let the simulation manager continue to the next step. If you wish to use prints in the Python code, they will appear in the terminal.