UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Físicas y Nanotecnología

Optimization of vehicular traffic flow in cities using complex networks techniques

Trabajo de integración curricular presentado como requisito para la obtención del título de Físico

Autor:

Juan Diego Vizcaíno Soriano

Tutor:

Mario Cosenza, PhD.

Cotutor:

Ramon Xulvi, PhD.

Urcuquí - Octubre,2024

Autoría

Yo, Vizcaíno Soriano Juan Diego, con cédula de identidad 1722103098, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor/a del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, Octubre, 2024.

[illegible]

Juan Diego Vizcaíno Soriano
CI: 1722103098

Autorización de publicación

Dedication

A mi madre, a mi padre y a la paradójica relación de un uno conformado por cinco (ahora 7)

Acknowledgements

A mis abuelos: Adalgiza, Guadalupe y Alfonso, sin ustedes nada de esto sería posible. Agradezco al profesor Mario Cosenza y al profesor Ramón Xulvi por ser los primeros en creer en un proyecto propuesto por mí. Agradezco a los chicos de la 512: Jorge (pison), Lucho (elbobo), Sebas(cpoot4) y Francis(chickenpapi) por estar siempre para mí. Agradezco al drink team: Oscar(Oscarin), Jimmy(Jimmyboy) y Stefy(herrote) por demostrarme que sí es posible salir de loqueo 4 días seguidos y sobrevivir. Agradezco a Vero por las maratones de películas y por soportarme. A mi trío de enanas: Andrea(negris), Dianita(la grandota) y Monse(arrocito) por escucharme y reírse conmigo de todo, en las buenas y en las malas.

Resumen

Proponemos un modelo conceptualmente simple para estudiar el importante problema del tráfico vehicular en la ciudad de Quito. Específicamente, estudiamos el impacto de los ciclos de los semáforos en el tiempo de viaje promedio de los vehículos. Tratamos el sistema de tráfico como una red compleja e interconectada en lugar de unidades aisladas. Hemos implementado nuestro modelo en un código computacionalmente eficiente en Python. Nuestros hallazgos indican que las configuraciones de los semáforos influyen significativamente en los tiempos de viaje, con ciertas configuraciones que reducen el tiempo de viaje para algunos agentes mientras lo aumentan para otros. Al considerar las interacciones entre los semáforos, mostramos que se puede lograr una configuración óptima que minimiza el tiempo de viaje promedio para la mayoría de los ciudadanos. Nuestros resultados muestran que, aunque algunas configuraciones benefician a la mayoría, también pueden afectar negativamente a una minoría de usuarios, especialmente en términos de tiempo de viaje para trayectos cortos. También demostramos que es posible mantener un tiempo de viaje promedio relativamente constante para varios agentes, lo cual es crucial para optimizar el flujo de tráfico sin afectar significativamente a ningún grupo específico. Aunque nos hemos centrado en las configuraciones de calles y semáforos de Quito, nuestro modelo puede aplicarse a otras ciudades de nuestro país y del mundo.

Keywords: Sistemas complejos, Modelos basados en agentes ,Flujo de tráfico , Comportamiento colectivo.

Abstract

We propose a conceptually simple model to study the important problem of vehicular traffic in the city of Quito. Specifically, we study the impact of traffic light cycles on the average travel time of vehicles. We treat the traffic system as an interconnected complex network rather than isolated units. We have implemented our model in a computationally efficient code in Python. Our findings indicate that traffic light configurations significantly influence travel times, with certain setups reducing travel time for some agents while increasing it for others. By considering interactions among traffic lights, we show that an optimal configuration that minimizes average travel time for the majority of citizens can be achieved. Our results show that, while some configurations benefit the majority, they may also negatively impact a minority of users, particularly in terms of travel time for short trips. We also demonstrate that it is possible to maintain a relatively constant average travel time for several agents, which is crucial for optimizing traffic flow without greatly affecting any specific group. Although we have focused on the street and traffic lights configurations of Quito, our model can be applied to other cities in our country and the world.

Keywords: Complex systems, Agent-based models, Traffic flow, Collective behavior.

Contents

List of Figures

Chapter 1

Introduction

Vehicular traffic in big cities is a complex and dynamic phenomenon that plays a crucial role in the daily lives of urban dwellers. As cities grow and develop, the movement of vehicles—ranging from personal cars and motorcycles to public buses and commercial trucks—becomes an essential aspect of urban infrastructure and functionality. Efficient traffic flow is vital for economic productivity, public safety, and overall quality of life.

Advancements in technology and data analysis have provided new tools for monitoring and optimizing traffic flow. Innovations such as intelligent transportation systems, real-time traffic monitoring, and adaptive traffic signal control have been employed to enhance the efficiency of urban traffic networks. Furthermore, the rise of ride-sharing services and the potential for autonomous vehicles promise to revolutionize how cities manage and experience vehicular traffic.

Despite these advancements, many cities still grapple with significant traffic congestion and its associated problems. Addressing these challenges requires a multi-faceted approach that includes infrastructure improvements, regulatory measures, and public engagement.

In recent years, traffic simulation models have provided an important tool to analyze and understand the complex behavior of vehicular traffic in urban environments. According to Paruchuri et al. (2002)[1], traffic simulation models can be broadly categorized into two types: macroscopic and microscopic. Macroscopic models view traffic flow in a broad sense, using aggregate variables such as traffic density, flow, and speed. These models are suitable for analyzing large-scale traffic patterns and regional transportation networks. Microscopic models, on the other hand, simulate the behavior of individual vehicles in great detail, providing insights into interactions and dynamics at a local level.

A common belief among urban inhabitants is that the city size is correlated with traffic congestion. However, recent studies have shown that this idea is due to a misconception. In the context of Ecuador, according to a Big Data company called INRIX, in 2018 the city of Guayaquil, with a surface area of just 344.5 square kilometers[2], was ranked as the 21st most congested city in the world[3]. This places Guayaquil in the same category as cities like Moscow and Sao Paulo, whose surface areas are of $2,511$ and $1,521$ square kilometers, respectively[4]. Further more, by 2019 the city of Quito, with a surface area of $372,4$ square kilometers[5], was listed in this ranking as the 18th most congested city in the world[6]. In contrast, cities with similar surface areas were not even ranked among the 100 most congested cities. For example, Munich, with a surface area of $310,4km^2$, and Dresden, with $328,8km^2$, were ranked 110th and 162nd, respectively, in the Traffic Index ranking published by navigation company TomTom in 2018[7].This challenges the assumption that traffic congestion mostly appears in big cities.

In 2019, the mobility department of Quito used video cameras installed at traffic lights at key locations to collect information about traffic flow density at various congested zones throughout the city[*]. Based on this data, several sources such as, Diario El Comercio[8], Infobae[9]and Gestion Digital[10], confirmed that by 2019 the average velocity in rush hours in Quito along key locations like Av. Gaspar de Villarroel, was between 13km/h and 17.7 km/h , while during other hours

[*]The documents provided by the Secretaría de Movilidad de Quito did not include a detailed explanation of the specific methodology used to collect or process the traffic density data. They only mention that the data was collected hourly using video cameras installed at traffic lights

of the day the average velocity in other urban zones reached a maximum of just 29km/h. By 2022, it was estimated that drivers in Quito lost around 170 hours per year in traffic jams. This means that, in average, people lost 7 days of their life in 2022 due to traffic congestion[8].

The huge problem of the vehicular traffic in the city of Quito, with its many social, economic, and political consequences, constitutes the main motivation for the realization of the present thesis.

In this thesis, we propose a model to optimize the average velocity of vehicular traffic flow by analyzing the influence of different configurations of the traffic light cycles in the streets of the city of Quito. We implement our model in a computationally efficient code in Python programming language. Our model aims to reduce the average travel velocity of the vehicles along the city by finding optimal traffic light cycles configurations.

1.1 Research problem

Motivated by the urgent need for solutions to the traffic problem in big cities and specifically in the city of Quito, this thesis proposes and implements a model to simulate and analyze vehicle movement on the city's street network. Employing Agent-Based Modeling and complex network concepts, we aim to study the impact of various traffic light configurations on traffic flow and analyze their influence on average travel times for vehicles in Quito.

1.2 Objectives

1.2.1 General objective

To accurately simulate the traffic flow in streets of the city of Quito and investigate the impact of traffic light configurations on the travel time of vehicles.

1.2.2 Specific objectives

1. To develop a conceptually simple and computationally efficient model to study the vehicular traffic in Quito, based on natural representations of the behavior of an agent moving along the city streets.
2. To study the influence of different traffic light configurations on the total average travel time of vehicles.
3. To search for the existence of an optimal configuration of traffic lights cycles that decreases the average travel time in the streets of Quito.

1.3 Overview

In Chapter 2, we introduce the concept of traffic flow in cities as a complex system and the implementation of Agent-Based modeling as the preferred computational method for our study. Chapter 3 contains the model that we propose and the main contributions of our work. We develop, step by step, our own theoretical approach towards a general probabilistic equation that allows us to find the optimal path between two intersections on the street map of the city of Quito. We implement the computational algorithm for the equation in Python programming language and illustrate it with examples. This Chapter also presents our ideas to incorporate the traffic lights cycles and the waiting times of vehicles at traffic lights. Several simulations that of the full model are performed and results are shown and interpreted. Finally, Chapter 4 contains the Conclusions on the present Thesis.

Chapter 2

Theoretical framework

The idea of cities as complex systems was first proposed in 1961 by a young writer called Jane Jacobs[11]. Her ideas set the intellectual agenda for the next 50 years of Urban Sciences[12].

In recent years, physics and other sciences have created the general concept of complex systems to describe a diversity of natural and artificial systems. A complex system is a set of interacting elements whose collective behavior cannot be derived from the knowledge of the properties of the isolated elements. The collective behavior is said to emerge from the interactions between the components, without any external influence or design. These systems commonly exhibit two properties: self-organization and emergence. The first one corresponds to the display of organization without the application of an external organizing principle or rule. The second is the manifestation of properties that are not present in the constituents of the system nor can be described by the superposition of their properties. Complex systems typically have a heterogeneous structure i.e., a large number of agents capable of interacting with each other at different scales and able to interact with the environment, leading to the evolution of their properties in time. These agents are not necessarily identical; they can be diverse and interact by different rules. The interactions between agents can be characterized as links forming a complex network, where the topology of the connectivity is neither uniform nor trivial[13].

To understand the structure of this type of systems we use complex networks (see Figure 2.1), which are a simplified representation of the system made up of nodes that are connected by links or edges. The nodes can represent individuals or groups , while connections represent relations between them, such as internet connections, neuron connections, protein interactions, etc.[14]

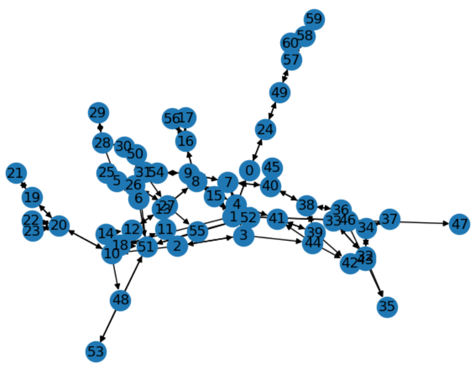


Figure 2.1: A complex network is a model for a set of dynamical elements represented as nodes with their mutual interactions as links.

The dynamics of many complex systems can be simulated through a computational approach called Agent Based Model. This computational method offers a natural representation of the system and allows to study the behavior of emergent phenomena in complex systems[15].

In general, creating an accurate model for a system is quite difficult. While statistical models are useful, they are based on assumptions that do not fully capture the complete dynamics of the system. For example, in traffic systems, the behaviors of drivers vary dramatically, with potential irrational behavior, subjective choices, and complex psychology[14]. Therefore no model can perfectly predict their behavior. Now, depending on its intended use –for long-term prediction for example- a basic, simplified model may be adequate[16].

In this context, there are two main types of simulation approaches: the macro level and micro level models. The macro level involves modeling the general aspects of the system like the average speed of all vehicles on road or the vehicle density in a given area. This types of model focus on studying the dynamics of the traffic flow and are principally used to deal with the tactical problems of strategic planning[17]. A micro level model, on the other hand, involves modeling each of the vehicles characteristics like its length ,width of or even the the psychological features of the drivers. These models treat the traffic flow movement at individual level. Indeed, the state and dynamics of the traffic flow is directly related to the interactions between each vehicle[18]. Substantial research has been published on the study of traffic flow optimization using this types of models.

For instance, Findler and Stapp (1992)[19] presented a macro level simulation on the control of traffic signals with distributed processors situated at street intersections. Each processor operates an identical expert system that communicates with adjacent processors, allowing for real-time data exchange and collaborative decision-making. They proposed this architecture to address the limitations of traditional traffic signal control methods, which often treat intersections in isolation and fail to adapt to dynamic traffic conditions. By leveraging a distributed system, the authors aimed to enhance the responsiveness and efficiency of traffic management, ultimately leading to improved traffic flow and reduced congestion. They consider that the interdependencies between adjacent intersections, is essential for effective traffic management. The proposed system incorporates multidimensional learning programs that optimize both the hierarchy of control rules and the parameters embedded within them. This allows for the adaptation of traffic signal strategies to varying conditions, such as time of day, day of the week, and seasonal variations, thereby improving the overall performance of the traffic network.

A different approach was proposed by Paruchuri et al. (2002)[1]. They presented a micro level model that explores the feasibility of using multi-agent simulation to model unorganized traffic, where drivers may not adhere to traffic rules or where rules are absent. The authors argue that traditional traffic simulators, which focus on organized traffic with well-defined rules, fail to capture the dynamic and unpredictable nature of unorganized traffic. The investigation highlights the importance of dynamic agents, capable of reacting to changing traffic conditions and anticipating future events. It emphasizes the need for a communication model that reflects real-world driver interactions, including visual cues and estimations of speed. The study is led by simulation where each vehicle is represented as an agent, capable of sensing its environment and making decisions to achieve its goal – reaching a destination. To reduce the computational overhead, the authors introduce a centralized agent and a blackboard concept, where the road itself acts as an agent, sensing vehicle positions and providing information to individual agents. The authors also discuss the psychological factors that influence overtaking decisions, introducing concepts like a confidence factor and a rush factor. Finally, they address the challenges of real-time simulation, particularly the need to balance individual agent processing with the dynamic nature of traffic.

While micro and macro models are the most common approaches to studying traffic flow, other models offer novel perspectives. For instance, Tan et al. (1995)[20] described a fuzzy-logic controller that aimed to determine the time that the traffic-light should stay in a certain state (red or green), before switching to the next state[21]. They state that conventional traffic-lights systems, whether fixed-time or vehicle-actuated, lack the flexibility to adapt to dynamic traffic conditions. They argue that a fuzzy-logic approach, mimicking human reasoning and decision-making, can provide a more efficient and responsive solution. Their investigation details the design and implementation of a fuzzy-logic traffic-lights controller for an isolated four-way intersection. The controller utilizes two fuzzy input variables: the density of traffic on the arrival side and the density of traffic on the queueing side. The output fuzzy variable is the extension time needed for the green light on the arrival side. Finally, the authors conclude that the fuzzy-logic approach offers significant advantages over conventional traffic-lights control systems. Its ability to handle fuzzy information, adapt to changing traffic conditions, and mimic human reasoning makes it a promising solution for improving traffic efficiency and reducing congestion in urban environments. The paper highlights the potential of fuzzy-logic as a powerful tool for developing intelligent systems that can effectively address complex real-world problems.

Here years later than Tan et al, Thorpe[22 23] and Wiering[21] proposed a different novel approach. They investigated traffic light control optimization using reinforcement learning (RL) algorithms. Their approach was based on the development of a reinforcement learning-based traffic light controller that learns the expected waiting times of cars at each intersection and sets the traffic lights to green for the configuration that minimizes the overall waiting time for all cars. The effectiveness of this approach is demonstrated through extensive simulations conducted in the Green Light District (GLD) simulator, a custom-built environment allowing for the creation of various city infrastructures, traffic patterns, and controller evaluations. The experimental results show that TC-1 significantly outperforms other controllers, reducing average waiting times by more than 25% in crowded traffic scenarios. This improvement is attributed to TC-1's ability to dynamically adapt to changing traffic conditions, unlike traditional controllers that rely on fixed settings.

A more innovative approach was carried on by Taale et al. (1998)[24]. They proposed the implementation of evolutionary algorithms (EAs), typically used in biological sciences, to optimize traffic light controllers, focusing on the use of an evolution strategy (ES) for finding optimal traffic light schedules to evolve a traffic light controller for a single simulated intersection. The paper concludes by emphasizing the potential of EAs for optimizing traffic light control, highlighting the need for further research to generalize the model, incorporate exception handling, and conduct more extensive simulations

In this thesis, we present a significantly different approach from the ones described previously. For instance, the model developed in this thesis is based on a real street map (specifically, the street map of Quito), while the studies mentioned before were based on simulated streets. Our approach allows us to study the average behavior of agents along the roads in Quito in a more realistic way. Additionally, our model was developed with the aim of replicating the individual driver behavior along the road in the most basic manner. Our model then focuses on preserving a few fundamental micro-characteristics of the system, such as the psychology of the driver, while also aiming to study macro-characteristics of the system, such as the average speed along the roads. In this context, our model can be categorized as a hybrid model.

Chapter 3

The Model

In this chapter, we explain how the model we use to simulate traffic flow in the city of Quito was constructed. First, we settle an ideal case to explain the logic behind the derivation of the mathematical expression for modeling the behaviour of a person when traveling from an initial location towards its destination. Secondly, we focus on constructing a general equation that allows an agent to find a path between two locations in the street network of Quito. Finally, we study how the configurations of traffic lights affects the total simulation time.

3.1 Developing the decision factor

To accurately emulate the general behaviors of a driver moving along the city streets, we first have to understand the most basic variables that influence the decision-making process of the driver. Therefore, we present a simple ideal case where a driver must decide between two given paths to arrive at his/her destination. For simplicity, the driver can solely move forward, and neither traffic lights nor velocities are considered in this first scenario.

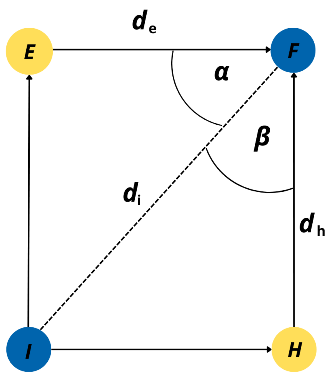


Figure 3.1: In this diagram, node I represents the driver's current position and node F represents the final destination. The arrows between nodes represent streets, and the nodes are the intersections between these streets. This means that travel between nodes is only possible if there is an arrow linking them. The direction of the travel is dictated by the direction of the arrows. The dotted line between I and F represents the "imaginary" direct route and the shortest euclidean distance between these two points. In this scenario, the angles α and β represent how much each of the possible paths are deviating from the "direct path"(the dotted line).

In this context, it seems logical to assume the driver would pick the path that he/she considers to be the closest to his/her destination. By definition, the shortest route between two points in space is a straight line i.e., the shortest euclidean distance between the two locations. This is represented in Figure 3.1 by the dotted line linking I (current position) and F(destination). However, when we drive along real cities, commonly we do not find a direct street connecting us to our destination. Thus, in this scenario, the driver has to choose between the two given routes towards F. The first one is given by going from I to E, and then moving towards F. The second route follows the arrow linking I with H, then continues to F [*].

Since we have stated that we will not consider any common traffic interactions like speed or traffic lights, the driver must consider only two variables: Firstly, which path deviates the least from the imaginary straight line that joins their current position and their destination (deviation angle). This deviation is represented in Figure 3.1 by the angles α and β, each of them representing how much each of the neighbors of I deviates from the optimal imaginary route. Secondly, the driver should consider which of the neighbors of I will drive them closer to F, i.e., which of the neighbors has the smallest Euclidean distance towards the destination node. The distances between E and H with respect to F are represented in Figure 3.1 by d_e and d_h, respectively.

We have simplified the decision problem to two variables: the deviation angle of each possible path and its euclidean distance from the destination. Our task now, is to develop a mathematical expression in the form $f_j(d_j, \theta_j)$ that tells the driver which is the best option for him/her to take. We also need to consider the idea of certain variables carrying different weight in our decisions. For example, when choosing a restaurant we might consider the cost and the location. In some cases, the cost might have a grater *weight* when making our decision compared to the location. Similarly, we need to assign these *weights* to the function inputs, the Euclidean distance (d_i) and the deviation angles (θ_i).

Based on this assumptions we derived the following equation:

$$f_j(d_j, \theta_j) = W_d\left(\frac{d_j}{d_I}\right) + W_\theta\left(\frac{\theta_j}{\theta_{max}}\right). \tag{3.1}$$

Let's now explain the logic behind this equation. First, note that the index j iterates over all neighbors of the current node. For example, in Figure 3.1, j index over E and H. Thus, when $j = E$ the function looks like $f_E(d_E, \theta_E)$ where $d_E = d_e$ and $\theta_E = \alpha$. Equally, when $j = H$ we have $f_H(d_H, \theta_H)$ where $d_H = d_h$ and $\theta_H = \beta$.

Second, in Equation 3.1, W_d and W_θ represent the weights assigned to distance and deviation angle, respectively. Since these two factors together determine the decision, their weights sum to one: $W_d + W_\theta = 1$. For instance, if the driver prioritizes minimizing distance regardless of deviation, then $W_d > W_\theta$. Conversely, if the driver prioritizes minimizing deviation regardless of distance, then $W_d < W_\theta$.

Thirdly, d_I represents the Euclidean distance between points I and F. Meanwhile, θ_{max} denotes the maximum acceptable deviation angle from the straight line connecting I and F. For instance, θ_{max} sets a limit on how much the driver can deviate from the direct path to the destination.

In this context, if we compute the value of f_j for the driver's current position, i.e., $j = I$, we find that $f_j = 1$. This implies that the driver has not moved from their initial position. Now, if we consider a neighbor's Euclidean distance, we observe an interesting pattern: when this distance is greater than the distance between the current location and the destination ($d_j > d_I$), the function's value exceeds one. Conversely, if the Euclidean distance of the neighbor is smaller($d_j < d_I$) the function's value tends towards zero.

This signifies that, as the value of f_j decreases, then the driver is getting closer to their destination. Conversely, if f_j is greater than one, it indicates that the driver is moving further away from their current position (in the next section we will explain why is important that this value becomes grater than one and not less). This relationship also holds true for the deviation angle: the greater the deviation angle of a neighbor, the further the driver is moving away from the destination [†].

Finally, the normalization of both variables d_j and θ_j also helps us to eliminates units and consolidates the information into a single numerical value. This normalization simplifies the representation, transforming a two-variable term into a more concise and manageable form.

[*]Nodes linked by an arrow are said to be neighbours. For instance, E and H are neighbors of I and F but I and F are not neighbours between them.

[†]A perceptive reader might have notice that while the example above, 45 degrees would be a great θmax, in real-world scenarios we may necessitate a max deviation up to 360 degrees.

3.2 Developing a probabilistic equation

In the previous section we formulated a mathematical expression to capture the decision-making process. This equation provides a numerical value that encapsulates the contribution of each variable in making a decision. Our goal now is to expand this approach into a probabilistic model to better reflect real-world driving scenarios. This approach makes sense, as in real-life scenario, drivers rarely make absolute, sequential decisions; instead, their choices are often probabilistic [‡]. Following this, our probabilistic approach looks as follows :

$$F_j(d_j,\theta_j) = \frac{f(d_j,\theta_j)}{\sum_n f(d_n,\theta_n)}, \tag{3.2}$$

In Equation 3.2, the index j represents the neighboring node to which we are considering moving, and the summation index n iterates over all the possible neighbors of the current position. Thus, when computing equation 3.2 for each neighbour, the resulting values will tell us the probability of moving to each of the neighbors respectively.

Now, as stated in the previous section, when considering nodes with f_j greater than one a potential issue arises: the contribution of these nodes in Equation 3.2 become greater, resulting in a higher probability of moving towards them, which is not the desired outcome. To address this issue, we work with the inverse of Equation 3.1. This approach ensures that nodes farther away from our initial position have a smaller contribution and, consequently, a smaller probability of being chosen as the next move and at the same time, nodes that are closer to the destination (a lower f_j) become a more probable choice. The general probabilistic equation then looks as :

$$F(d,\theta) = \frac{\left(\frac{1}{f(d_j,\theta_j)}\right)^{\delta}}{\sum_n \left(\frac{1}{f(d_n,\theta_n)}\right)^{\delta}} = \frac{f(d_j,\theta_j)^{-\delta}}{\sum_n f(d_n,\theta_n)^{-\delta}}, \tag{3.3}$$

where the exponent δ controls the *contribution* of each of the weights in the decision factor. This parameter can be seen as a smoothing effect as the weight associated with the larger ratio becomes more dominant. By substituting Eq. (3.1) in Eq. (3.3), we get the general probabilistic equation

$$F(d,\theta) = \frac{\left(W_d\left(\frac{d_j}{d_I}\right) + W_\theta\left(\frac{\theta_j}{\theta_{max}}\right)\right)^{-\delta}}{\sum_n \left(W_d\left(\frac{d_n}{d_I}\right) + W_\theta\left(\frac{\theta_n}{\theta_{max}}\right)\right)^{-\delta}}. \tag{3.4}$$

3.2.1 Testing the probabilistic equation

In this section, we will detail how we obtain the street map of Quito and calculated the Euclidean distances and deviation angles between nodes using the OSMNX and Geopy python packages. Then, we test the accuracy of Eq. (3.4).

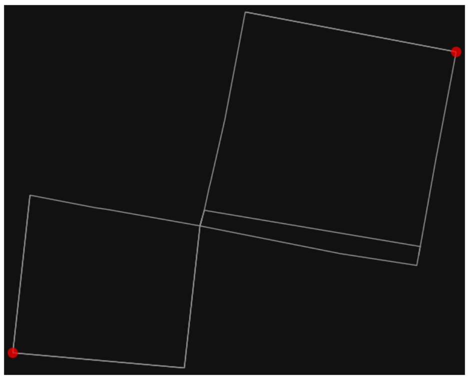
To begin with, we imported the street map of Quito at the intersection of Avenida Río Coca and Isla Merchana Street (Figure 3.2.a), which closely resembles our ideal case. We achieved this using the OSMnx Python package [§], which interacts with three public web APIs[¶]. Specifically, OSMnx leverages the OpenStreetMap Nominatim API, the Overpass API, and the Google Maps Elevation API. By leveraging the features and graph modules provided by OSMnx, users can download OpenStreetMap data from the Overpass API using various methods, including latitude-longitude point coordinates, address, bounding box, bounding polygon/multipolygon, or place name[26]. This allows us to upload the street map of any location as a directed graph, where nodes represent intersections between streets and edges represent the roads

[‡]For example, when encountering traffic jam on our usual route home the probability of following that same route would decrease. Thus, we would likely seek an alternate path that gets us to our destination more quickly

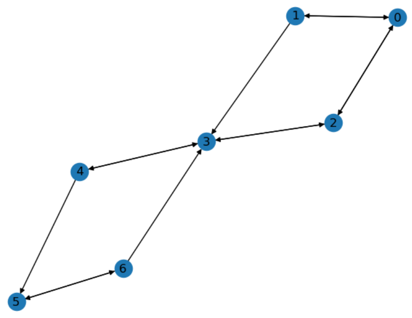
[§]See: https://osmnx.readthedocs.io/en/stable/

[¶]A Web API is an application programming interface for the Web that allows one software application to interact with another[25]. In simple terms, it is like a waiter in a restaurant. It takes requests and brings back responses connecting the intersections. Each node is assigned a unique ID, enabling the identification of each intersection on the network. Additionally, each edge in the graph contains information about the streets it represents, including their lengths, directions, street names, and sometimes even the speed limit allowed for the streets.

Then, to obtain the real values of the Euclidean distances between each neighbor and the destination node, we developed a Python function using the Geopy package[‖]. This function allows us to determine the Euclidean distance between two points by providing the latitude-longitude point coordinates of each location. The latitude-longitude point coordinates were provided by the Osmnx package, as mentioned before, and compared and verified using Google Maps. Furthermore, the deviation angle between the nodes was calculated by developing another Python function using Geopy and Osmnx packages [**]. This function first computes the geodesic[††] between the initial position and the destination. Subsequently, by calculating each geodesic of the neighbors in relation to the destination, it measures how much each neighbor deviates from the geodesic between the current position and the destination.



(a)



(b)

Figure 3.2: **a)** The street network map on the intersection of Avenida Río Coca and Isla Merchana Street. The upper right node represents node 0 and the current position of the agent, while the lower left red dot corresponds to node 5, which is the agent final destination. **b)** Graph representation of the street network map. In this representation, the arrows denote the streets, with the direction of the arrow indicating the possible direction of travel. Specifically, a double arrow indicates that travel is possible in both directions

After this, we define the values of W_d and W_θ. For the purpose of this thesis, we will assume that the primary objective of every driver is to get as close to their destination as possible. Consequently, we will assume that a driver would prefer to travel to a node that brings him/her closer to his/her destination rather than deviating from it, implying that $W_d > W_\theta$. Therefore, we will set $W_d = 0.60$ and $W_\theta = 0.40$ accordingly.

Next, as discussed in the previous section, the value of θ_{max} will determine the maximum deviation from the destination that we will allow. Considering that in reality, there may be cases where a deviation of more than 270 degrees might be possible, we will set $\theta_{max} = 2\pi$. Finally, the smoothing parameter will be set to a minimum value that enables us to study the contribution of each variable to the final probability. Therefore, we will set $\delta = 2$. Then, the general equation takes the form:

[‖]Geopy is a python pacakge that makes it easy to locate the coordinates of addresses, cities, countries, and landmarks across the globe using third-party geocoders and other data sources.[27]

[**]All the python functions developed for this thesis can be found at github.com/JuanDiego-28

[††]A geodesic is the shortest distance between any two points on a sphere[28]

$$F(d_j, \theta_j) = \frac{\left(0.60\left(\frac{d_j}{d_I}\right) + 0.40\left(\frac{\theta_j}{2\pi}\right)\right)^{-2}}{\sum\limits_{n}\left(0.60\left(\frac{d_n}{d_I}\right) + 0.40\left(\frac{\theta_n}{2\pi}\right)\right)^{-2}}. \tag{3.5}$$

We can now evaluate the accuracy of Equation (3.5) by testing it on the imported map. In this initial simulation, the agent's[‡‡] goal is to travel from node 0 to node 5 (see Figure 3.2.b) by identifying the best neighboring node (either node 1 or node 2) to reach the destination. To achieve this, we need to calculate the Euclidean distances between the starting node 0 and the destination node 5, as well as the Euclidean distances from each neighbor (node 1 and node 2) to node 5. These distances are denoted by d_0, d_1, and d_2, respectively. Using the Python functions explained earlier, we obtain the following values: $d_0 = 343.57m$, $d_1 = 288.36m$, and $d_2 = 267.19m$. Next, we calculate the deviation angles of each neighbor, which are given by $\theta_1 = 15.59°$ and $\theta_2 = 15.13°$. With these values, we can now compute the results of $F(d_1, \theta_1)$ for node 1 and $F(d_2, \theta_2)$ for node 2 to determine which node is more likely for our agent to travel to. Replacing the values in Eq. (3.5), we get for node 1,

$$F_1(d_1, \theta_1) = \frac{\left(0.60\left(\frac{d_1}{d_0}\right) + 0.40\left(\frac{\theta_1}{2\pi}\right)\right)^{-2}}{\sum\limits_{n=1,2}\left(0.60\left(\frac{d_n}{d_0}\right) + 0.40\left(\frac{\theta_n}{2\pi}\right)\right)^{-2}} = 0.4627 \tag{3.6}$$

and for node 2,

$$F_2(d_2, \theta_2) = \frac{\left(0.60\left(\frac{d_2}{d_0}\right) + 0.40\left(\frac{\theta_2}{2\pi}\right)\right)^{-2}}{\sum\limits_{n=1,2}\left(0.60\left(\frac{d_n}{d_0}\right) + 0.40\left(\frac{\theta_n}{2\pi}\right)\right)^{-2}} = 0.5372 \tag{3.7}$$

According to the values we have just calculated, our agent is more likely to move to node 2[§§] with a probability of $F_2 = 0.5372$. Once at node 2, the agent repeats the same process it employed in node 0. First, since we are now in node 2, the value of d_I is updated from $d_i = d_0$, to $d_i = d_2 = 267.19m$, the Euclidean distance from node 2 to node 5. Next, the agent has to choose to which of his neighbours it should move to get closer to its destination. For instance, the agent has two options: (i) to move back to node 0 [¶¶] (the two-way arrow pointing from 2 to 0 in Figure 3.2.b implies that both ways travel is possible), or (ii) to move forward to node 3. Let us then compute the values of $F(d, \theta)$ for each of the possibilities. For going back to node *0* we have

$$F(d_0, \theta_0) = \frac{f(d_0, \theta_0)^{-2}}{\sum\limits_{j=0,3} f(d_j, \theta_j)^{-2}} = 0.248, \tag{3.8}$$

and for node 3 we get

$$F(d_3, \theta_3) = \frac{f(d_3, \theta_3)^{-2}}{\sum\limits_{j=0,3} f(d_j, \theta_j)^{-2}} = 0.752. \tag{3.9}$$

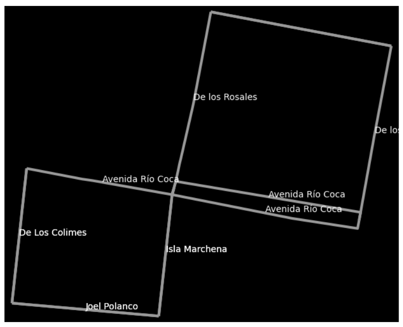
From the above calculations, we conclude that the agent is most likely move to node 3. Upon reaching node 3, the agent can move forward to node 4 or go back to node 2 (there is a two–way arrow pointing from node 3 to node 2). At

[‡‡]From now on, when we say "agent," we will be referring to the driver, and vice versa.

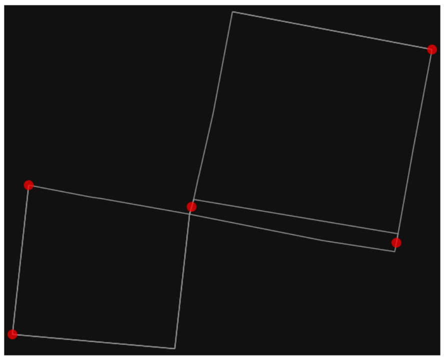
[§§]With this result, we have validated the claim made in Section 3.2. Since node 2 has a shorter Euclidean distance to node 5 compared to node 1, the probability of moving towards node 2 increases.

[¶¶]A valid query in this part is: what if, for some reason, the probability of returning to node 0 were higher? This question is addressed in the following section.

node 3, the agent repeats again the same procedure and decides to move forward to node 4 with $F_4 = 0.781$. Finally, in node 4 the agent moves towards its destination node 5, with $F_5 = 0.995$.



(a)



(b)

Figure 3.3: **a)** Map of the 200m radius surrounding the intersection of Avenida Río Coca and Isla Merchana Street. **b)** Path that the agent finds after the decision-making process. When comparing this graphic with Figure 3.2.a, each new red dot represents a node chosen by the agent to reach node 5. For instance, the red dot at the bottom right is node 2, the red dot at the center is node 3, and the upper one at the left is node 4.

We have shown that our probabilistic equation works quite accurately. We can check this by looking at the probabilities we have calculated. For instance, when the agent arrived at node 3, it had the option to either go back to node 2 or keep moving forward to node 4. The resulting probabilities confirmed what we expected: the agent chooses the direction that he/she *believes* will get him/her closer to the destination (Fig. (3.3)).

3.3 General path finder algorithm

In this Section, we address the two main challenges we encountered while implementing our proposed general equation Eq. (3.5) and the development of a general path finder algorithm.

For instance, the first challenge we faced was that some agents got trapped in routes with no exit. This happened when an agent found a node with a smaller Euclidean distance towards the destination than the actual node it was supposed to follow and this mistaken node turned to had no exit. Secondly, as noted in the previous section, some agents became trapped in a loop between two nodes. This occurred when, after computing the decision equation for its neighbors, an agent returned to a previous node, only to be directed back to its original node, thus creating a cycle.

To exemplify this situations we have developed Figure 3.4. Here, the agent must pass through node 0 to reach its destination, node F. However, node 0 is less likely to be chosen due to its greater distance from node F compared to nodes 1 and 2. If the agent does choose node 0, it will likely return to node I as it is closer to node F than node 3, creating a cycle. On the other hand, choosing node 1 or 2 leads to a dead-end node. Consequently, the agent gets stuck in a loop between the chosen node and the initial position, just like before. From this analysis, it is easy to notice that both problems are caused due to the probabilities are not correctly updated when moving backwards to the previous position; instead, they remain unchanged.

To address this, we propose to introduce a penalty factor (ρ) to the output of Eq. (3.5) when an agent returns to a previously visited node. This penalty factor decreases the probability that the agent chooses that node again, effectively reducing the chances of getting stuck in an infinite loop. To better understand this strategy, consider the following Algorithm 1 that presents the key steps involved in this process [***]:

[***]The code snippets of this algorithm can be found at appendix A

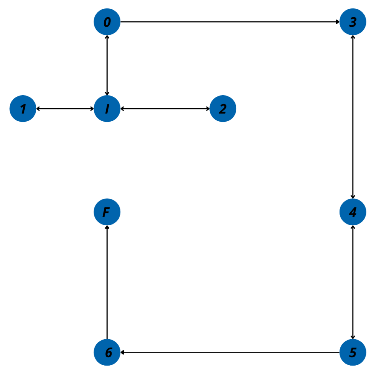


Figure 3.4: The only way to reach node F is by passing through node 0; however, the distance from node 0 to F is greater than the Euclidean distance from the agent's current position I to F. Additionally, both node 1 and node 2 are dead-end nodes.

Algorithm 1 Path Finder algorithm

```
Require: $i, f \leftarrow$ initial node , destination node
 1: while $i \neq f$ do
 2:     for n in neighbours of i do
 3:         $F_n(d_n, \theta_n)$
 4:         if n in previously visited then
 5:             $F_n(d_n, \theta_n) \leftarrow F_n(d_n, \theta_n) \times \rho$
 6:         end if
 7:     end for
 8:     choose n with highest $F_n(d_n, \theta_n)$
 9:     move to n
10:     $i \leftarrow n$
11: end while
```

This method allows the agents to learn from their experiences and adjust their behavior. For instance, consider Figure 3.3. In this scenario, the agent has three options: node 0, node 1, and node 2. The probabilities of going to each of the neighbors are given by F_1, F_2, and F_0. Additionally, let's assume $F_1 > F_2 > F_0$, with node 1 being the best option and node 0 the least probable one.

In the first iteration of the algorithm presented above, the agent chooses to move towards node 1. At node 1, the agent computes the probabilities, and since the only neighbor of node 1 is node I, the agent goes back to it. Once at node I, the agent recomputes the probabilities of going to each of the neighbors. At this point, since the agent has already visited node 1, the probability of going towards it is updated to $F_1 = F_1 \times \rho$. Then, the probabilities now look like $F_2 > F_0 > F_1$. Thus, the agent now moves towards node 2.

At node 2, the same situation occurs as at node 1 - the only possible path is to return to node I. So, even though the agent comes from node I and the penalty factor is applied [†††], the agent still goes towards it. Again, at node I, the probabilities are recomputed and updated. Thus, F_2 becomes $F_2 = F_2 \times \rho$ and $F_0 > F_1 > F_2$.

Now, the agent can finally move towards node 0, where it computes the probabilities of going back to node I or move forward to node 3. Then, the probabilities are computed and updated to $F_I = F_I \times \rho$ and hence $F_3 > F_I$ allowing the agent to move towards its destination.

[†††]In this first iteration, the penalty factor is not applied to F_I since at that point it was the starting point. By the second iteration, we treat it as a visited node.

To test the efficiency of this algorithm, we uploaded the street map of a 1 km radius around the intersection of Los Shyris Avenue and Tomás de Berlanga Avenue (see Figure 3.5).

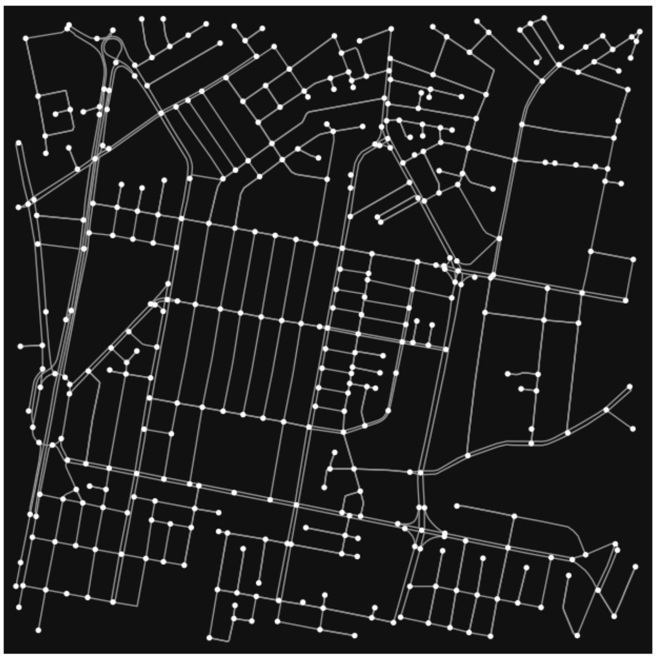
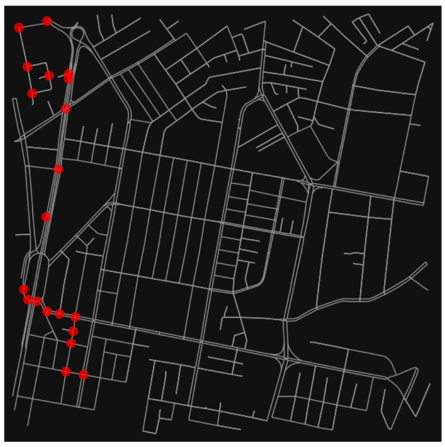
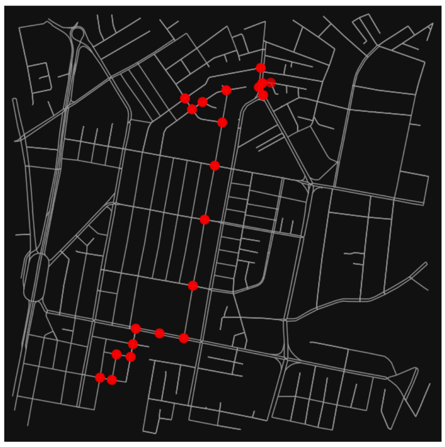


Figure 3.5: Representation of the street map network.

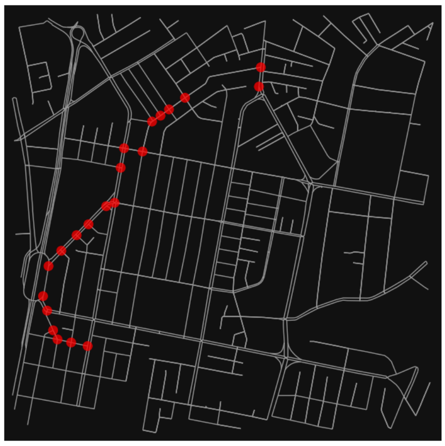
Along with some simulations we conducted on the imported map, we found that even after applying the penalty factor to some probabilities, they would still be higher than the probabilities of the node the agent must follow to achieve its destination. In such cases, what happened was that the agents looped between the nodes, updating the probabilities until the penalty factor had the desired impact. Eventually, all the agents were able to find paths between any random initial and destination nodes on every location of the road network. Below, we present some of our results.



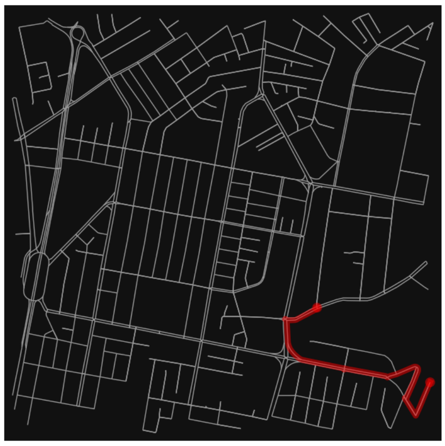
(a)



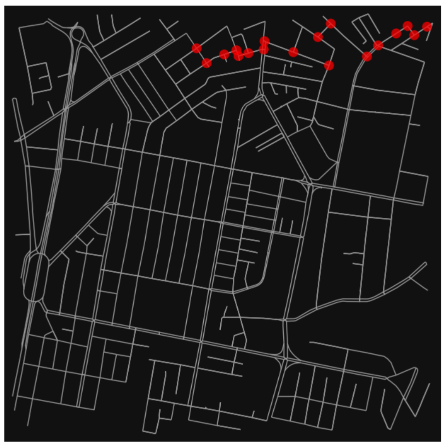
(b)



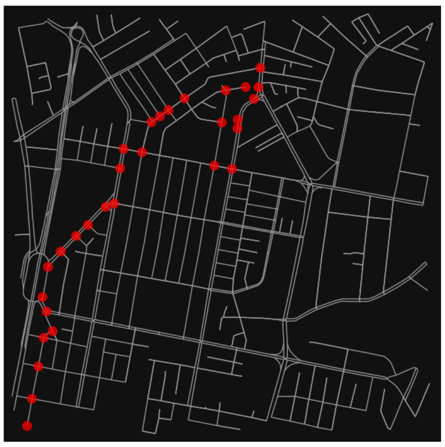
(c)



(d)



(e)



(f)

Figure 3.6: Different paths found by agents along the street network between random initial and destination nodes, after the implementation of Eq. (3.5) and the penalty factor.

3.4 Defining the waiting time and implementing the traffic lights

In this section, we explain the implementation of the waiting time of agents in front of traffic lights. To achieve this, we first had to identify and locate each of the traffic lights along the street network of Quito that we are working with (see Figure 3.5). It is important to recall that nodes in the network correspond to intersections between streets, which is precisely where traffic lights are located. As mentioned in Section 3.2.1, the OSMNX Python library provides a data frame with several pieces of information about the imported graph. Among the information it provides, we have a unique ID for each intersection (node), the names of the streets (edges) that form the intersection, and the geocoordinates of this intersection.

To locate these intersections, we used the data provided by the Mobility Department of Quito (MDQ). In the files provided by them, the location of the traffic lights was given by the names of the streets of the intersection where the lights were located. However, in many cases, the names of the streets provided by OSMNX differed in some grammatical aspects from the real names, such as capital letters or accent marks (like the tilde). Since a single edge could have more than 20 intersections with other streets and hence more than 20 nodes associated with that name, we developed a Python function using GeoPandas that allowed us to identify each node using their geocoordinates. To get the geocoordinates of the traffic lights, we used Google Maps [‡‡‡], and then, after uploading these values into our Python function, the function returned the ID of the node in the data frame, allowing us to locate each of the traffic lights in the graph we imported.

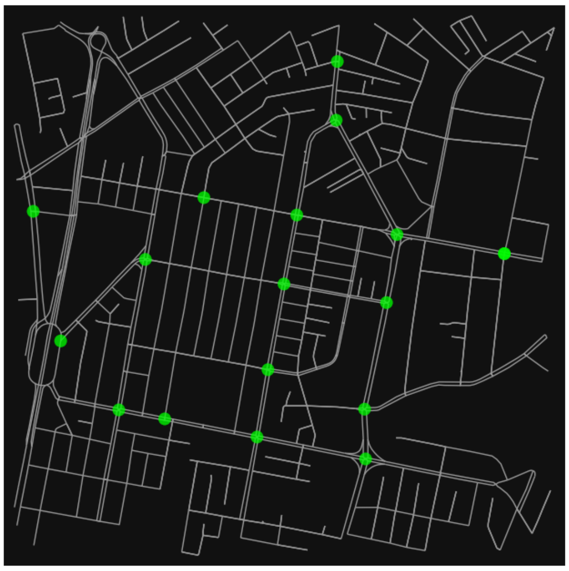


Figure 3.7: Here, the green dots represents the traffic lights along the steet network.

Now that we have located are the traffic lights in the map, we can establish the configurations of the traffic lights cycle in our simulation. First, we have to remember that, in real life, when we are approaching a traffic light we are not aware of the likelihood of encountering the light in the green or red state (for simplicity, in our simulation we will only focus on the red and green states). The traffic light is constantly repeating the cycle described by $\Delta_v \rightleftarrows \Delta_r$, where Δ_v represents the amount of time when the light is green and Δ_r is the amount of time when the light is red.

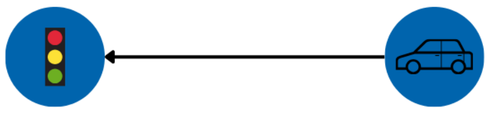
Now, to calculate the corresponding probabilities, we must remember that the traffic light cycle repeats over the total time τ that the agent spends reaching the traffic light. Thus, the probabilities of being in either at the red or green state depend entirely on time. Specifically, on the duration of the cycle in the red or green state and the time it takes for the agent to reach the traffic light.

[‡‡‡]We introduce the names in the files provided by the MDQ in google maps. Once we found the node we copy and paste the coordinates of this location in the python function.

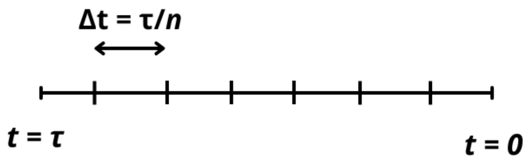
Let's begin by defining the probabilities of encountering the traffic light in the red state, denoted as p_r, and in the green state, denoted as p_v. These probabilities should be expressed in terms of Δ_r and Δ_v. Thus, the probabilities can be represented as follows:

$$p_v = \frac{\Delta_v}{\Delta_v + \Delta_r}, \qquad p_r = \frac{\Delta_r}{\Delta_r + \Delta_v}. \tag{3.10}$$

Secondly, as mentioned earlier, the probability of finding the traffic lights in any of the possible states also depends on the time it takes the agent to reach them (see Figure 3.8.a). Naively, we can discretize this total travel time and calculate the probabilities p_r and p_v for each sufficiently small n-time step. Essentially, we divide the total time τ into n smaller intervals of time Δ_t, such that $\Delta_t = \tau/n$ (see Figure 3.8.b).



(a)



(b)

Figure 3.8: **a)** A agent is moving from towards a traffic light. **b)** Total time τ for the agent to reach the traffic light is discretized into n smaller intervals Δ_t.

Then, we can define the probability of find traffic light at either red or green state for a given time interval Δ_t as

$$\Delta_t p_r + \Delta_t p_v = 1 \Rightarrow \Delta_t p_r = 1 - \Delta_t p_v \Rightarrow \Delta_t p_v = 1 - \Delta_t p_r \tag{3.11}$$

Furthermore, we now that for calculating the total probability of two events occurring consecutively, we have to multiply the probabilities of both events across time. Therefore, the probability of finding the red state at a given time τ is obtained by multiplying each probability along all n intervals as follows

$$(1 - \Delta_t p_v)(1 - \Delta_t p_v) \times \cdots \times (1 - \Delta_t p_v) = (1 - \Delta_t p_v)^n. \tag{3.12}$$

Therefore, with $P_r = \Delta_t p_r$ and for small enough Δ_t intervals, we can approximate P_r to

$$\begin{aligned} P_r &= \lim_{\Delta_t \to 0} (1 - \Delta_t p_v)^n \\ &= \lim_{n \to \infty} \left(1 - p_v \frac{\tau}{n}\right)^n \\ &= e^{-p_v \tau}. \end{aligned} \tag{3.13}$$

In the same way, the probability P_v that the agent finds the traffic light in the green state at a given time τ is

$$P_v = 1 - P_r. \tag{3.14}$$

Now, if the agent reaches the traffic light in the red state, he/she will have to wait for the traffic light to change to green. This *waiting time* is calculated as follows:

$$t_s = P_r(\Delta_r). \tag{3.15}$$

By incorporating the impact of traffic lights on the travel time of the agents, we can analyze the influence of different traffic light cycles on the overall simulation time and individual travel times. Table 3.1 presents the various cycles of Δ_v and Δ_r that will be used in the simulations at section in Section 3.6

Simulation	Frequencies	
	$\Delta_v(s)$	$\Delta_r(s)$
0	60	60
1	45	60
2	57	38
3	60	50
4	20	60
5	57	43
6	75	65
7	120	60
8	80	120
9	90	30

Table 3.1: This table presents a selection of traffic light cycle configurations employed in each of the ten simulations. These configurations were derived from data provided by the MDQ. The MDQ did not specify whether these cycles were updated based on real-time traffic flow, time of day, or through the intervention of an algorithm or departmental personnel. Consequently, we utilized a subset of these values and calculated average cycle times for our analysis

3.5 Average velocity

To determine the velocities of the agents along the roads, let us consider what happens when a vehicle approaches an intersection in real life. A responsible driver will reduce the vehicle's velocity and after exiting the intersection, the driver will accelerate up to reach a constant velocity along the street, just to repeat this process in each intersection along his/her route. It is important to remember that the average velocity of a vehicle is also affected by other external factors, like the number of cars in front of it along a street. For example, on a 500m street, having 5 cars in front of you will likely result in a higher average velocity than having 10 or 15 cars in front on the same street.

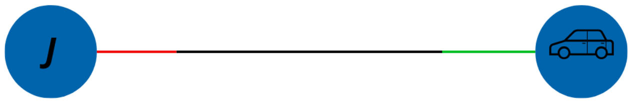


Figure 3.9: During the green interval, the driver accelerates until he/she reaches a constant speed along the street (black segment). Upon approaching a new intersection, the driver begins to decrease the car's velocity, as indicated by the red segment. In real life, this interval will vary depending on the length of the street, the amount of cars in front of the driver and the driving style of each driver.

Due to time limitations in the development of this work and to focus on the primary objective of demonstrating the influence of traffic lights on the total travel time of agents, we will simplify the modeling of the average velocity of the agents. This simplification consists of assigning an average velocity to each agent along the edges (streets) of the network based on the length of the edge they are traveling on. For instance, for streets less than 500m, the average velocity will be a random number between 13 km/h and 19 km/h, while on larger streets, it will be between 20 km/h and 29 km/h. These velocity ranges are derived from data provided by the MDQ, indicating that the average velocity in the area where the simulations were conducted ranges from a maximum of 29 km/h in the absence of traffic jams to a minimum of 10 km/h in the most congested scenarios[§§§].

[§§§]Due to limitations in the data collection methodology, such as the lack of specification for which types of vehicles each average velocity corresponds

3.6 Simulations

In this section we will explain how we conducted simulations to study the influence of traffic lights on the total average travel time of the agents. This will be done by breaking down the process of the simulation (see Figure 3.11) into clear steps providing a more detailed explanation of our methodology .

Before initiating the simulations, we assigned a different fixed starting and destination node for 10 different agents (see Figure 3.10). Each agent then determines the path between its initial position and destination using the general path finder algorithm (see Algorithm 1). In this instance, we assume that the objective of any driver in the city of Quito is to get as close as possible to their destination, prioritizing the Euclidean distance over the deviation angle. Therefore, the parameter values for equation 3.5 remain as $\delta = 2$, $W_d = 0.60$, and $W_\theta = 0.40$, as defined in section 3.2.1.

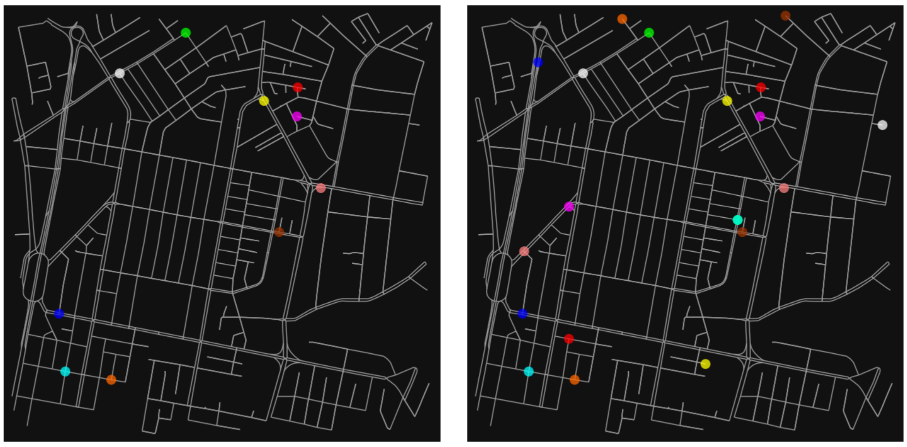

(a) (b)

Figure 3.10: (a) Initial position of each of the ten agents labeled by colors. (b) The new nodes represent the destination node of the agents. Each color represents a different agent.

Furthermore, we have defined the penalty factor to be $\rho = 0.40$. This value was selected to reduce the number of iterations between the loop of dead-end nodes and the correct nodes with higher Euclidean distance [¶¶]. Once the agents have found their paths, they store the ID of each of the nodes that conform their route.

Parameters	Values
δ	2
W_d	0.60
W_θ	0.40
ω	0.60
ρ	0.40

Table 3.2: Parameters values for simulating traffic flow in Quito.

After completing the previous step, the simulation starts by locating each of the 10 agents at their starting nodes at time $T = 0$. Then, each agent begins their journey towards their destination following the route they had found earlier.

to, we have decided to set up a range of values according to the distance of the streets, assuming that higher velocities can be achieved on larger streets. However, we have cross-validated these values with the INRIX data sets (see bibliography[3]) and confirmed their consistency.

[¶¶]Notably, when testing different values for the penalty factor, in some cases, the iterations exceeded 16 iterations for the factor to take the desired outcome. Therefore, aiming to reduce this number of iterations, we chose a value that drastically reduced this value to no more than 9 iterations.

Once the simulation has started, we track the agents using the ID of the nodes and the sequence in which they are visited. Based on the street length, we assign an average velocity to the agent for each street on its route. We determine the length of each edge (street) that conforms the route of the agent based on the ID of the node he/she comes from and the ID of the node he/she is going to. Every time an agent exits an edge and enters the next one, a new average velocity is assigned based on the length of the new street and stored by the agent. Once the average velocity is assigned, we calculate and store the time (t_i) it takes for each agent to travel along the edge.

Furthermore, whenever an agent is approaching a new intersection (node), we use the ID of the new node to identify if that node corresponds to a traffic light. If the node ID corresponds to a traffic light, we calculate the probability (P_r) of it being at the red state. To provide a clear decision boundary and eliminate ambiguity in the simulation, we introduce a threshold condition (ω) to determine if the traffic light is set as red or not (see table 3.2). Therefore, if $P_r > \omega$, the traffic light is in the red state. Otherwise, the traffic light is in the green state. Finally, if the traffic light is found at the red state, we calculate the waiting time (t_s) with equation 3.15 and add it to the total time (t_i) the agent takes to cross that edge. On the contrary, if the traffic light is found at the green state, the waiting time $t_s = 0$.

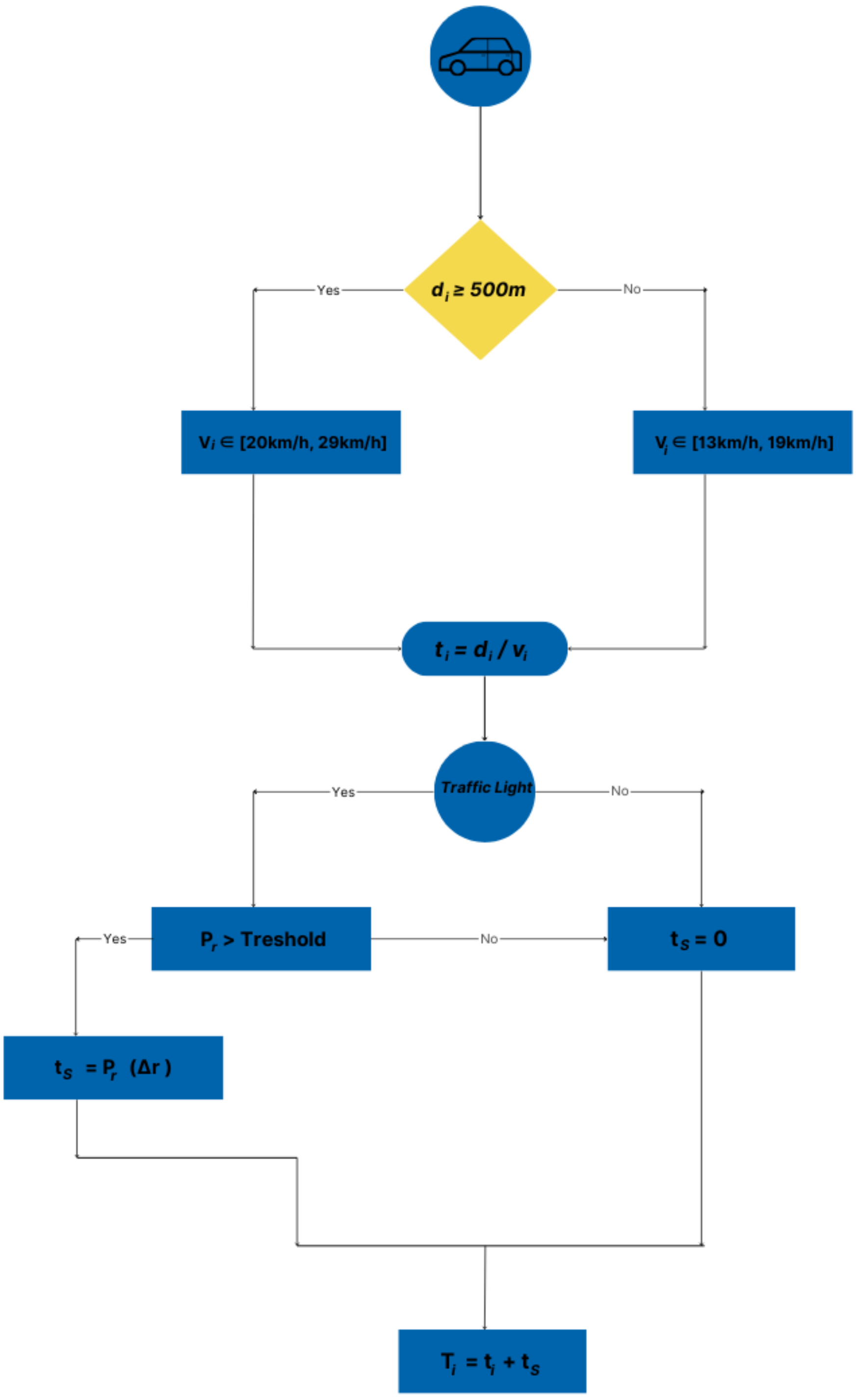


Figure 3.11: In this flow diagram, d_i is the distance of the edge where the agent is traveling, v_i is the assigned velocity, t_i is the average time it takes the agent to travel along the edge, t_s is the waiting time at traffic lights, and T is the total travel time of the agent. Note that for each agent, the value of T will be different since they all have different routes

As each agent stores the assigned velocities along all edges, the total time it takes for them to reach their destinations is only affected by how each traffic light cycle is defined in each simulation. Since the probability P_r depends on the values assigned to Δ_v and Δ_r, we are able to set different configurations of the traffic light cycles along the network for each traffic light based on the values presented in table 3.1. This allows us to analyze the influence of the traffic light cycles on the total simulation time and the individual travel times of each agent. Finally, note that the simulation ends once all agents have reached their corresponding destinations, meaning the simulation lasts as long as it takes the last agent to reach its destination node.

3.7 Results

The path found by each of the agents is presented below at figure 3.12

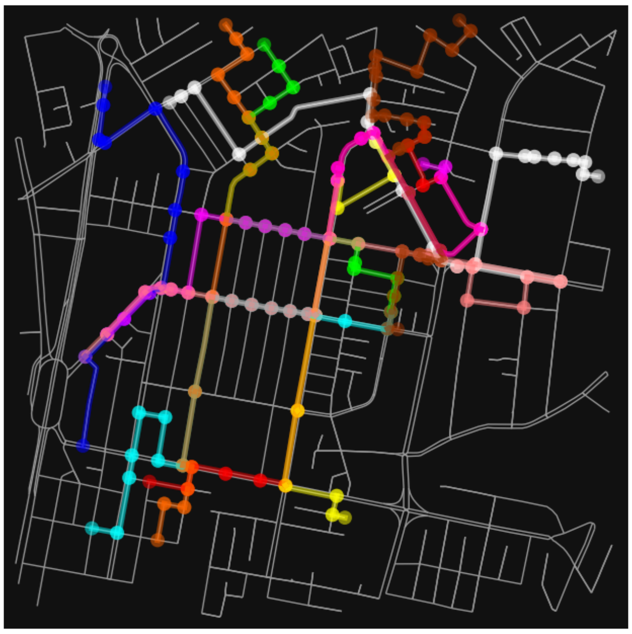


Figure 3.12: Here we present the paths each agent followed towards their destination. Each color represent each agent. Note that for some agent the colors superimpose between them due to they pass through the same nodes.

The results obtained are summarized in Table 3.3. This Table illustrates the average time calculated from 10 simulation runs over a given traffic light configurations. Then, the process was repeated for 10 different traffic light configurations.

Agents	**Traffic Lights Frequencies**									
	Freq0	**Freq1**	**Freq2**	**Freq3**	**Freq4**	**Freq5**	**Freq6**	**Freq7**	**Freq8**	**Freq9**
0 (white)	17:38	16:56	19:50	21:02	22:28	20:27	14:02	18:03	17:57	15:10
1 (red)	17:02	13:08	15:42	15:52	15:50	15:10	14:50	14:29	14:46	16:20
2 (green)	9:45	7:05	8:52	9:45	10:12	9:39	7:44	9:01	8:36	8:16
3 (blue)	9:08	9:56	10:09	10:34	10:57	10:41	8:02	9:54	9:08	8:25
4 (yellow)	10:19	7:42	7:47	7:46	7:33	7:36	9:09	7:42	7:32	10:14
5 (violet)	17:35	16:6	18:30	20:13	21:39	19:43	13:01	19:20	16:28	14:28
6 (turquoise)	13:42	12:35	16:40	17:50	18:54	16:38	10:10	16:22	14:20	11:14
7 (orange)	10:57	11:19	11:07	11:59	12:16	12:06	9:45	11:07	11:08	10:16
8 (pink)	17:22	13:32	20:35	21:38	22:31	19:02	12:48	15:39	16:22	14:26
9 (brown)	11:44	12:39	13:49	14:35	15:36	14:54	9:18	13:45	12:12	9:54
Simulation time	**17:38**	**16:56**	**20:35**	**21:38**	**22:31**	**20:27**	**14:50**	**18:03**	**17:57**	**16:20**

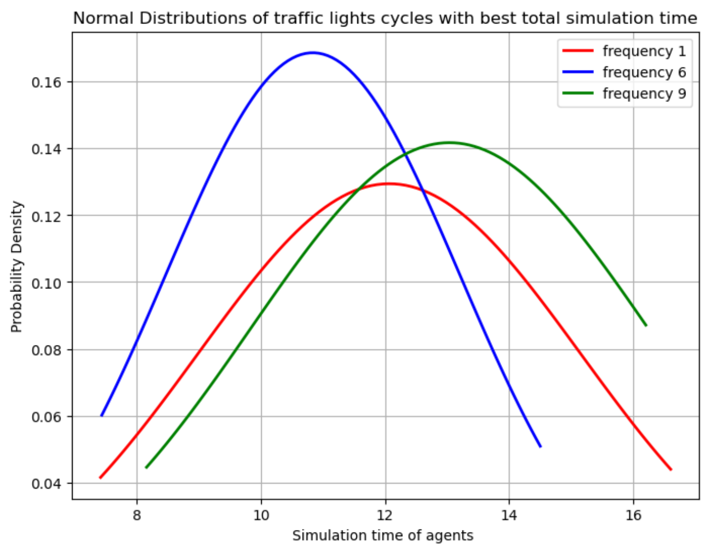
Table 3.3: Average travel time for each agent with different traffic light configuration cycles.

The results show that, for different configurations of traffic lights cycles, the travel time of each agent exhibits a significant average time difference.

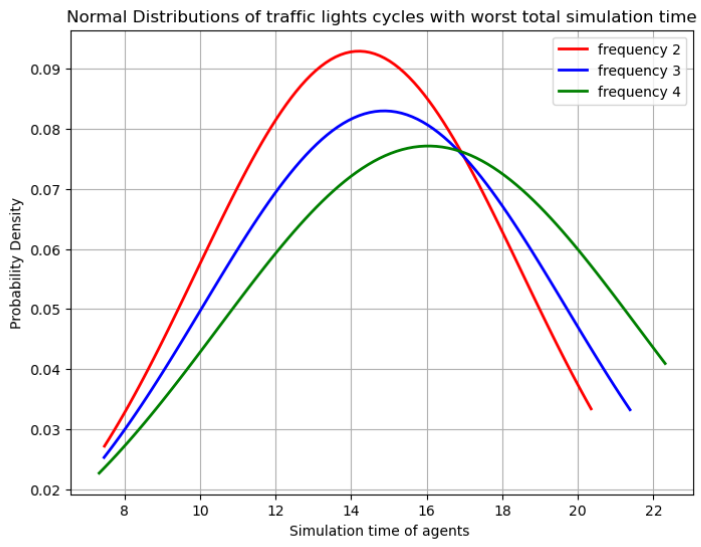
For instance, the best traffic light configuration for agent 0 is configuration 6, resulting in an average travel time of 14 minutes and 2 seconds. On the other hand, the worst configuration for agent 0 is configuration number 4, with a total travel time of 22 minutes and 28 seconds. If we compare those same configurations but of agent 4, we can see that in configuration 6 turns out to be the worst one with an average travel time of 9 minutes and 9 seconds whereas in configuration 4, it is 7 minutes and 33 seconds.

Let us now take a closer look at the total simulation time for each of the configurations. It is important to remember that the simulation time is considered finished once the last agent reached his/her destination. Therefore, the configuration cycle with the shortest total simulation time, will represent, on average, the best configuration for all agents. This, by no means guarantees that all agents will reduce their total travel time. For instance, upon reviewing Table 3.3, we can confirm that the best simulation time is achieved in configuration 6. By minute 14:02, all agents had already reached their destination node. What is striking about this table is that it confirms our previous statement. Specifically, with configuration 6, most of the agents showed a drastic reduction in their average travel time, while in contrast, agent 4 was the only one displaying an increase in his/her travel time.

Another aspect observed in Table 3.3 is that, if we consider configuration 9 it shows a more homogeneous time distribution than other configurations along the simulation. Interestingly , even though configuration 9 presents an increase in the average travel time with respect to configuration 6, it is still the second best average travel time achieved between the 10 configurations.



(a)



(b)

Figure 3.13: **(a)** Probability distributions comparing the three best configurations, and **(b)** the three worst ones.

Surprisingly, some configurations not only reduce the total travel time for certain agents but also they keep their average travel time relatively constant. For example, agent 4 maintained a relatively consistent average travel time across configurations 1 to 5, as well as configurations 7 and 8. Similarly, for agent 1, configurations 1 to 3 reduced the average time while keeping it somewhat constant.

Furthermore, in Figure 3.13, we can observe that for configuration 6, the variation in the total travel time of the agents does not spread as much as in the configurations with the worst total simulation time, such as configurations 2 or 3. For instance, the configuration that shows the most significant difference between the travel times of each agent is frequency 4, which also has the worst total simulation time. Notably, in Figure 3.13, even though these are the three configurations with the worst total simulation time, we can see that the three curves converge to a common simulation time. For instance, with a closer inspection of Table 3.3 and Figure 3.13, we can see that frequencies 2, 3, and 4, although they prejudiced the majority of the agents, also maintain a relatively constant total travel time between most of them.

Chapter 4

Conclusions

In this thesis, we have developed a conceptually simple probabilistic model to study the important problem of the vehicular traffic in the city of Quito. Specifically, our model considers how different configurations of the variable Freq can affect the average travel time of citizens in the city of Quito. Although we have focused on the street and traffic lights configurations of Quito, our model can be applied to other cities in our country and the world.

We have implemented our model into an efficient computational code in Python. It should be emphasized that the mathematical model and the computational programs have been developed by the author of the present thesis. Even though we used a simple approach, our model can provide important conclusions.

First, we have demonstrated that different traffic light configurations significantly impact the total travel time of drivers. For example, the results of our simulation show that configuration cycle 6 (see Table 3.3) substantially decreased the travel time for a subset of agents while significantly increasing the total time for others. This suggests that the configuration for each of the traffic lights along the city should not be considered as isolated systems.

Second, if we aim to optimize the traffic flow of the city we should consider it as a network that allows each of the traffic lights (nodes) to interact with one another. In this way, we can reach an optimal configuration time that achieves the lowest average travel time for the majority of the citizens.

Third, our results show that, although a given frequency configuration of traffic lights may highly benefit the majority of the agents, it may at the same time negatively affect some others. The total travel time can be reduced by several minutes for large travels, but it seems to increase some minutes for short travels.

Fourth, we have found that for given configurations, we can also maintain a relatively constant average time for several agents. This is an important result because the main goal is to reduce the average time of the agents while trying to affect the least other agents. From this finding, we can conclude that indeed, it is possible to maintain consistency in average travel time for more than one agent. This is crucial because reducing travel time does not necessarily means achieving higher velocities along the path, but rather maintaining a relatively constant average speed throughout the route for all the agents.

Finally, we are aware that to solve the problem of traffic jams in the city of Quito is not an easy task, by it surely requires an urgent solution. There are several approaches that the Secretaria de Movilidad could employ, among them upgrading public transportation, creating designated spaces for bicycles, implementing more efficient traffic rules, etc. However, it is important to recognize that these solutions are long-term initiatives that will take several years to be effectively implemented. However, the results of the present Thesis suggest that the installation of intelligent synchronized traffic lights can significantly reduce the time lost in traffic jams in a relatively short time and with a relatively lower cost of implementation.

Bibliography

[1] Paruchuri, P.; Pullalarevu, A. R.; Karlapalem, K. Multi agent simulation of unorganized traffic. 2002.

[2] Wikipedia, Guayaquil — Wikipedia, The Free Encyclopedia. http://es.wikipedia.org/w/index.php?title=Guayaquilol 2024; [Online; accessed 16-July-2024].

[3] INRIX, W. 2018 Global Traffic Scorecard. **2018**,

[4] Wikipedia, List of largest cities — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=List%20of%20largest%20citiesoldid=1233725611, 2024; [Online; accessed 16-July-2024].

[5] Wikipedia, Quito — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Quito&oldid=1232741503, 2024; [Online; accessed 16-July-2024].

[6] Scorecard, G. T. Global Traffic Score Card. *INRIX. Kirkland: Inrix Research* **2019**,

[7] BV, T. I. TomTom Traffic Index 2018. 2018; https://traffic-index-docs.s3-eu-west-1.amazonaws.com/TomTomTrafficIndex-Ranking-2018-full.pdf, Downloaded from: https://www.tomtom.com/traffic-index/ranking.

[8] Comercio, E. En Quito se viaja a 13 km/h en hora pico - El Comercio. https://www.elcomercio.com/actualidad/quito/trafico-vehicular-secretaria-movilidad-informe.html, 2019; Accessed: July 16, 2024.

[9] Loaiza, Y. Quito es la ciudad con mayor congestión vehicular de Ecuador y la tercera en Sudamérica. https://www.infobae.com/america/america-latina/2023/12/01/quito-es-la-ciudad-con-mayor-congestion-vehicular-de-ecuador-y-la-tercera-en-sudamerica/, 2023; Accessed: July 16, 2024.

[10] Gestión, R. El tráfico y la movilidad, el gran desafío de la nueva alcaldía de Quito. https://revistagestion.ec/analisis-sociedad/el-trafico-y-la-movilidad-el-gran-desafio-de-la-nueva-alcaldia-de-quito/, 2023; Accessed: July 16, 2024.

[11] Rowe, A. Jacobs: The Death and Life of Great American Cities. **1962**,

[12] Bettencourt, L. M. Introduction to urban science: evidence and theory of cities as complex systems. **2021**,

[13] Boccara, N. *Modelling Complex Systems*; Springer, 2016.

[14] Bazghandi, A. Techniques, advantages and problems of agent based modeling for traffic simulation. *International Journal of Computer Science Issues (IJCSI)* **2012**, *9*, 115.

[15] Burguillo, J. *Complex Networks*; Editorial 1, 2018; pp 35–56.

[16] Meyn, S. *Control techniques for complex networks*; Cambridge University Press, 2008.

[17] Buisson, C.; Lebacque, J.; Lesort, J. STRADA, a discretized macroscopic model of vehicular traffic flow in complex networks based on the Godunov scheme. 1996.

[18] others,, *et al.* Macro-micro simulation of traffic flow. *IFAC Proceedings Volumes* **2006**, *39*, 351–356.

[19] Findler, N. V.; Stapp, J. Distributed approach to optimized control of street traffic signals. *Journal of Transportation Engineering* **1992**, *118*, 99–110.

[20] Khalid, M. Intelligent traffic lights control by fuzzy logic. *Malaysian Journal of Computer Science* **1996**, *9*, 29–35.

[21] Wiering, M.; Vreeken, J.; Van Veenen, J.; Koopman, A. Simulation and optimization of traffic in a city. 2004.

[22] Thorpe, T. L.; Anderson, C. W. Tra c light control using sarsa with three state representations. *Technical report, Citeseer* **1996**,

[23] Thorpe, T. Vehicle traffic light control using sarsa. *MS tesis, Department of Computer Science, Colorado State University* **1997**,

[24] Taale, H.; Bäck, T.; Preuss, M.; Eiben, A.; De Graaf, J.; Schippers, C. Optimizing traffic light controllers by means of evolutionary algorithms. 1998.

[25] W3Schools, Web APIs. 2024; `https://www.w3schools.com/js/js_api_intro.asp`.

[26] Boeing, G. Modeling and Analyzing Urban Networks and Amenities with OSMnx. **2024**,

[27] GeoPy, GeoPy Documentation. 2023; `https://geopy.readthedocs.io/en/stable/`, Accessed: 2024-09-26.

[28] Encyclopaedia Britannica, Geodesic. 2023; `https://www.britannica.com/science/geodesic`, Accessed: 2024-09-26.

Appendices

Appendix A

Python program

The following code is made in python. The general decision equation and the path finder algorithm. The complete code and modules developed in this project can be found at : github.com/JuanDiego-28

```python
"""
Created on Dic 2023
@author: jdiego
"""

import networkx as nx
import numpy as np
import osmnx as ox
import functions


def path_finder(initial_node, final_node , G ):

    """
    This functions simulates an agent thar looks for the best path to follow from one node to another
    """

    # get dictionary with geocordinates of each of the nodes
    coordiantes = functions.get_geo_coordinates(G)

    node_i = initial_node

    # to store the routes:
    route= {}

    path = []

    # bug list
    bug_list = []
    # no exit list
    no_exit_list = []

    # loop to finde the path:
    while node_i != final_node:

        # store all nodes the agent pass through:
        path.append(node_i)

        # measure the angle and distnac between the node_i and the destination node
        angle_i = functions.get_angle( coordiantes[ node_i ], coordiantes[ final_node] )
        dist_i = functions.get_distance( coordiantes[ node_i ], coordiantes[ final_node] )

        # get the neighbours of the node. The neighbours are directed nodes
        neighbors = list(G.neighbors(node_i))

        # get probality of pick a given neighbour
        factors = {}

        for node in neighbors :
            # measure angle and dist of neighbors towards final node
            angle_neighbor = functions.get_angle( coordiantes[ node ], coordiantes[ final_node] )
            dist_neighbor = functions.get_distance( coordiantes[ node ], coordiantes[ final_node] )

            # calculate the probability of picking a node
            factors[node] = functions.get_factor(dist_neighbor , angle_neighbor , dist_i , angle_i , 0.

        # get probabilities
        proba_dic = functions.get_probability(factors)


        #print(" i am at node  ", node_i, " my options are : ", proba_dic)

        # list of probable nodes_id:
        list_proba = list(proba_dic.keys())

        # contador de veces que paso por un nodo
        repeticions = path.count(node_i)

        for node in list_proba:
            counter = path.count(node)
            if node in path:
                proba_dic[node] = proba_dic[node]*0.4
                if counter > 2:
                    proba_dic[node] = proba_dic[node]*(0.4**counter)
        # if i have pass more than two times by a node
        if repeticions > 2:

            # sotre node_i in wich bug is produced
            bug_node = node_i
            bug_list.append(node_i)
            #print("bugg en el nodo : " , bug_node )

            # count how many times a bug node has been pass through.
            counter = bug_list.count(node_i)

            if counter >= 3:
                vecinos_i = list(G.neighbors(node_i))
                for vecino in vecinos_i:
                    if vecino in  bug_list:
                        route[bug_node][vecino] = 0.0
                        proba_dic[vecino] = route[bug_node][vecino]
                        #print("me quedo en el nodo: " , node_i, "acualice probabiliades: ", proba_dic)
                        #print(route)
                #break

        # check for node with no exit (i.e no neighbors)
        if len(neighbors) == 0:
            # store in route

            no_exit_node = node_i
            route[no_exit_node] = {}
            no_exit_list.append(no_exit_node)

            # update to last position
            node_i= path[-2]

            # check neihbours
            vecino_j = list(G.neighbors(node_i))

            # update probability of going back
            for vecino in vecino_j:
                if vecino in no_exit_list:
                    route[node_i][no_exit_node] = 0.0
                    proba_dic = route[node_i]
                    #print("volvi al nodo: ", node_i, "se acutliazo las lista de probilidades a : " ,pr
            #break

        # check if the node has no posible routes to the destination
        no_options = all( options == 0 for options in proba_dic.values())
        if no_options:
            position = path.index(node_i)
            no_options_node = node_i
            no_exit_list.append(node_i)
            #print("ninguno de los caminos posibles es correcto")
            #print("regreso al nodo", path[position - 1 ])
            node_i =  path[position - 1 ]
            route[node_i][no_options_node] = 0.0
            proba_dic = route[node_i]

            # check there is no not exit routes
            vecino_z = vecino_j = list(G.neighbors(node_i))
            for vecino in vecino_z:
                if vecino in no_exit_list:
                    position_i = path.index(node_i)
                    node_j = path[position_i -1]
                    #print("debo regresar al node " , node_j, " no debo ir a ", node_i)
                    route[node_j][node_i] = 0.0
                    proba_dic = route[node_j]
                    print(proba_dic)
            #break

        # stop simulation if bug
        if repeticions > 15:
            print(" ERROR: el nodo " , node_i ," se repite mas de 15 veces")
            break

        # add to route:
        route[node_i] = proba_dic

        # ge the node id with max probability
        node_id = max(proba_dic , key= proba_dic.get)

        # get the max probability
        Proba = proba_dic[node_id]

        #print(" i am at node: " , node_id)

        # update position
        node_i = node_id

    route[final_node] = {final_node:1.0}

    return route, path
```