UNIVERSIDAD DE INVESTIGACIÓN DE TECNOLOGÍA EXPERIMENTAL YACHAY

Escuela de Ciencias Matemáticas y Computacionales

TÍTULO: LARGE SCALE TRAFFIC FLOW SIMULATION USING A DISTRIBUTED SYSTEM

Trabajo de integración curricular presentado como requisito para la obtención del título de Ingeniero en Tecnologías de la Información

Autor:
Clavijo Herrera Mauro Anibal

Tutor:
Ph.D. Pineda Arias Israel Gustavo

Urcuquí, Junio 2021

Urcuquí, 23 de junio de 2021

SECRETARÍA GENERAL
(Vicerrectorado Académico/Cancillería)
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2021-00017-AD

A los 23 días del mes de junio de 2021, a las 14:00 horas, de manera virtual mediante videoconferencia, y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa	Dr. IZA PAREDES, CRISTHIAN RENE , Ph.D.
Miembro No Tutor	Dr. MANZANILLA MORILLO, RAUL , Ph.D.
Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.

El(la) señor(ita) estudiante **CLAVIJO HERRERA, MAURO ANIBAL**, con cédula de identidad No. **0302695176**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, realiza a través de videoconferencia, la sustentación de su trabajo de titulación denominado: **Large scale traffic flow simulation using a distributed system.** , previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.

Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación a través de videoconferencia, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:

Tipo	Docente	Calificación
Miembro Tribunal De Defensa	Dr. MANZANILLA MORILLO, RAUL , Ph.D.	9,5
Presidente Tribunal De Defensa	Dr. IZA PAREDES, CRISTHIAN RENE , Ph.D.	9,0
Tutor	Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.	10,0

Lo que da un promedio de: **9.5 (Nueve punto Cinco)**, sobre 10 (diez), equivalente a: **APROBADO**

Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.

Certifico que *en cumplimiento del Decreto Ejecutivo 1017 de 16 de marzo de 2020, la defensa de trabajo de titulación (o examen de grado modalidad teórico práctica) se realizó vía virtual, por lo que las firmas de los miembros del Tribunal de Defensa de Grado, constan en forma digital.*

CLAVIJO HERRERA, MAURO ANIBAL
Estudiante

Firmado electrónicamente por:
MAURO ANIBAL CLAVIJO HERRERA

Dr. IZA PAREDES, CRISTHIAN RENE , Ph.D.
Presidente Tribunal de Defensa

Firmado electrónicamente por:
CRISTHIAN RENE IZA PAREDES

Dr. PINEDA ARIAS, ISRAEL GUSTAVO , Ph.D.
Tutor

Firmado electrónicamente por:
ISRAEL GUSTAVO PINEDA ARIAS

Dr. MANZANILLA MORILLO, RAUL , Ph.D.
Miembro No Tutor

Firmado electrónicamente por:
RAUL MANZANILLA

TORRES MONTALVÁN, TATIANA BEATRIZ
Secretario Ad-hoc

TATIANA BEATRIZ TORRES MONTALVAN
Firmado digitalmente por TATIANA BEATRIZ TORRES MONTALVAN
Fecha: 2021.06.28 21:52:48 -05'00'

Autoría

Yo, **Mauro Anibal Clavijo Herrera**, con cédula de identidad **0302695176**, 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 autor del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, Febrero del 2021.

Mauro Anibal Clavijo Herrera
CI: 0302695176

Autorización de publicación

Yo, **Mauro Anibal Clavijo Herrera**, con cédula de identidad **0302695176**, cedo a la Universidad de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Urcuquí, Febrero del 2021.

Mauro Anibal Clavijo Herrera
CI: 0302695176

Dedication

"To my immediate family: My father Enrique, my mother Blanca, my sister Angie and my grandparents. Without their unconditional love, support, patience and guidance this achievement could not have been possible. To Domitila, I know that you will always protect me from heaven.

To my excellent instructors whose perseverance and knowledge prepared me for any adversities that will arise during my career and life.

To my friends and fellow students specifically to Karen, Andres, and the PPP group. I thank you all for being part of this unforgettable journey. As always I wish you all the best in your future endeavours."

Acknowledgments

First of all, I want to acknowledge Israel Pineda, Ph.D. Without his advice and motivation, this project could have never been developed. His support during the implementation and writing of this project helped me to improve my research abilities that will be very useful for future projects. His dedication during this project made this achievement possible.

I would like to acknowledge this university, which provided me with a quality education and laid a strong foundation for a brighter tomorrow.

I also would like to acknowledge my friends who gave me ideas and recommendations during the development of this work.

Finally, I want to acknowledge my family for their guidance and unconditional love that accompanied me during this stage of my life.

Resumen

El aumento de la congestión vehicular en las grandes ciudades, la evolución constante de los sistemas informáticos y el desarrollo de sistemas de transporte inteligentes han dado lugar a simulaciones de flujo de tráfico que representan escenarios de tráfico de la vida real en un entorno de software. Estas simulaciones ayudan a mejorar el flujo del tráfico real mediante el análisis de datos, para predecir la congestión en calles o carreteras concurridas, y evitar inconvenientes como accidentes y atascos.

Simulation of Urban Mobility (SUMO) es una de las herramientas de simulación vehicular más populares. Este software es un paquete de tráfico microscópico y de código abierto que realiza simulaciones de grandes redes de carreteras. Sin embargo, estas simulaciones son computacionalmente costosas debido a problemas en tamaños de escenarios y al número de vehículos. La cantidad de vehículos en un escenario puede llevar a tiempos de procesamiento inmanejables. La necesidad de más computación produce requisitos más exigentes de características de hardware. Se implementan varios enfoques que utilizan técnicas paralelas o distribuidas para aumentar el rendimiento de la simulación, lo que permite al usuario experimentar con casos grandes.

Message Passing Interface (MPI) es un estándar de comunicación para enviar y recibir mensajes entre diferentes procesos que se ejecutan simultáneamente para realizar tareas paralelas o distribuidas. Las implementaciones de este estándar proporcionan varias funciones que permiten la comunicación entre procesos y permiten el paso de mensajes entre ellos. Las principales ventajas de MPI son su portabilidad para sistemas distribuidos y su amplia utilización debido a la optimización del hardware.

Este trabajo propone un algoritmo de partición de escenarios de simulacion y gestión de fronteras para controlar el modelo de flujo de tráfico. Estas ideas se implementan y ejecutan en grandes escenarios de SUMO utilizando un sistema distribuido. El algoritmo de partición utiliza un enfoque de espacio uniforme. Este sistema se implementa mediante técnicas MPI para comunicarse con diferentes nodos. Comparamos los resultados y el rendimiento de la simulación distribuida con los resultados de una simulación de un solo nodo. El desempeño de este enfoque se evalúa por el tiempo requerido para ejecutar la simulación. Los resultados comparativos demostraron que el tiempo necesario para calcular un paso de simulación disminuye. Sin embargo, las operaciones de sincronización y control incluyen una sobrecarga significativa en el tiempo total de simulación.

Palabras Clave: SUMO, simulacion distribuida, MPI, TraCI.

Abstract

The increased traffic congestion in big cities, the constant evolution of computer systems, and the development of Intelligent Transportation Systems (ITS) have resulted in traffic flow simulations representing real-life traffic scenarios in a software environment. These simulations help improve actual traffic flow using data analysis to predict congestion in crowded streets or highways and avoid inconveniences such as accidents and traffic jams.

Simulation of Urban Mobility (SUMO) is one of the most popular simulation tools. This software is a non-proprietary traffic package that performs simulations of large road networks. However, these simulations are computationally expensive due to large problem spaces in the number of vehicles. The number of vehicles in a scenario can lead to unmanageable processing times. The need for more computation produces higher demanding requirements of hardware characteristics. Few approaches using parallel or distributed techniques have been implemented to increase the simulation performance enabling the user to experiment with significant cases.

Message Passing Interface(MPI) is a communication standard to send and receive messages between different processes running simultaneously to perform parallel or distributed tasks. The implementations of this standard provide several functions that enable interprocess communication and allow message passing between them. The main advantages of MPI are its portability for distributed systems and its vast utilization due to hardware optimization.

This work proposes a network partition algorithm and border management to control the traffic flow model. These ideas are implemented and executed in large SUMO scenarios using a distributed system. The partition algorithm uses a uniform space approach. This system is implemented using MPI techniques to communicate with different nodes. We compare the results and performance of the distributed simulation with the results of a single-core simulation. The performance of this approach is evaluated by the time required to execute the simulation. Comparative results demonstrated that the time required to compute a simulation step decreases. However, synchronization and control operations include a significant overhead to the overall simulation time.

Keywords: SUMO, distributed simulation, MPI, TraCI

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Traffic congestion is one of the most challenging problems a city faces, with significant repercussions to its citizens. These scenarios can lead to accidents, delays, or other inconveniences with vehicles that circulate streets or highways in large urban areas. Traffic simulation can be a powerful tool to simulate real traffic scenarios in a city. These simulations predict traffic jams and avoid dangerous traffic situations. However, the number of vehicles in a congestion simulation can be vast, leading to computationally expensive simulations.

Several types of traffic simulations have been created and are currently studied. Various created traffic simulators perform macroscopical and microscopical simulations. The first one is based on the complete road flow using different constraints, and the last one on simulating each vehicle as an autonomous agent that possesses its behavior based on predefined metrics [7]. However, in this project, a microscopic simulator called SUMO [8] will perform large-scale simulations of traffic scenarios using a geographic partition method and several computation nodes to distribute the workload.

Simulation of Urban Mobility (SUMO) is a traffic simulation software designed during 2001 at the German Aerospace Center (DLR, German acronym). SUMO is an open-source traffic simulator that contains several applications enabling the preparation and performing simulations on real traffic scenarios [8]. This tool uses a network file that can be created using netedit or converted from an existing map with netconvert. These are some of the SUMO tools, among others. A route file also needs to be generated, which will contain some characteristics such as type, departure times, and predefined routes for all the vehicles in the simulation. Some car-following models simulate the driver behavior and avoid accidents by considering another vehicle within the simulation. The microscopical approach that SUMO implements considers each vehicle as an individual agent and declares several variables that simulate the traffic scenario.

Microscopic simulations tend to be computationally expensive because we represent each vehicle as an individual agent. We need to calculate several characteristics (i.e., position, speed, behavior) of each vehicle per time step. This high computation demand increases the hardware requirements for large scenarios and leads to longer execution times.

This time can grow exponentially as the simulation scenarios increase.

This project proposes a solution for this problem by performing a distributed traffic simulation using SUMO, Traffic Control Interface (TraCI), and a Message Passing Interface (MPI) library to increase large scenarios performance. The solution enables to run scenarios that would typically use a single machine in several machines. TraCI enables manipulating the simulation in "real-time" and adding/removing vehicles sent between nodes using MPI as a communication tool. The proposed system is compared to a standard simulation in terms of time performance.

This project uses a uniform space partition algorithm to divide the network and route scenarios into similar size partitions, as suggested by Acosta et al. [6]. Then, each node receives a partition. It will run a SUMO instance using said partition and interact with TraCI to calculate vehicular behavior and send/receive vehicles through an MPI communicator.

1.2 Problem Statement

Traffic scenarios are a crucial subject considered in urban planning as this can lead to congestion situations and inefficient road constructions. Traffic congestion is a widely researched area as the efforts aim to optimize route planning and avoid accident scenes [9][10]. These scenarios are often studied, taking advantage of the available technology using real-life scenarios and replicating them in a software-oriented simulation. A traffic simulator is software that renders a network (i.e., a city scenario) and simulates vehicular and pedestrian behavior using predefined variables such as traffic flow models and others.

Performing a traffic simulation is computationally expensive due to the number of computations required to calculate the characteristic of the vehicles at each step of the simulation. A distributed system can be an effective solution to the mentioned problem as this will allow running the computations in several nodes that will work together to process the calculations.

One of the motivations for using MPI techniques within a distributed system relies on portability and compatibility with different hardware architectures. It also provides a secure and easy message passing infrastructure that allows communication between different system nodes.

1.3 Objectives

1.3.1 General Objective

Design and implement a distributed system for traffic flow simulation using communication techniques and traffic control interface to manipulate vehicles in real-time during the execution of a SUMO scenario.

1.3.2 Specific Objectives

- Use a space partition algorithm to divide a network and vehicles inside a delimited area.

- Implement a distributed system to run simulations in SUMO using Traffic Control Interface (TraCI) to control objects in the simulation and MPI techniques to communicate nodes.
- Analyze different SUMO simulations using the proposed implementation.

Chapter 2

Theoretical Framework

This chapter presents the necessary concepts to understand this work, an introduction, a detailed explanation of the fundamentals in traffic simulations, the SUMO simulator interface, and components that help create traffic flow scenarios simulating them with user-created characteristics. It also explains the MPI communication techniques that enable us to create a message-passing system within the development of this project.

2.1 Traffic Simulation

The need for mobility during human activities, such as economic or social, has become essential nowadays. As the urban and rural zones have grown, so have the distance and the time required for transportation between two locations. The augmentation in vehicles that circulate the world every year results in this time increment [11]. Furthermore, the emergence of electric and self-driving cars [12] has enabled the development of models representing real-life traffic scenarios using computational applications that simulate said scenarios and even help us predict traffic.

The first traffic flow model was presented by Bruce Greenshields in 1935 [13], and since then, several studies developed models and simulation tools. Nowadays, long-term planning and short-term predictions, based on actual traffic data, employ simulation frameworks. In the future, the models and simulation tools may develop to support adaptive cruise control, dynamic traffic management, and other characteristics [1]. Figure 2.1 depicts the evolution of traffic flow models.

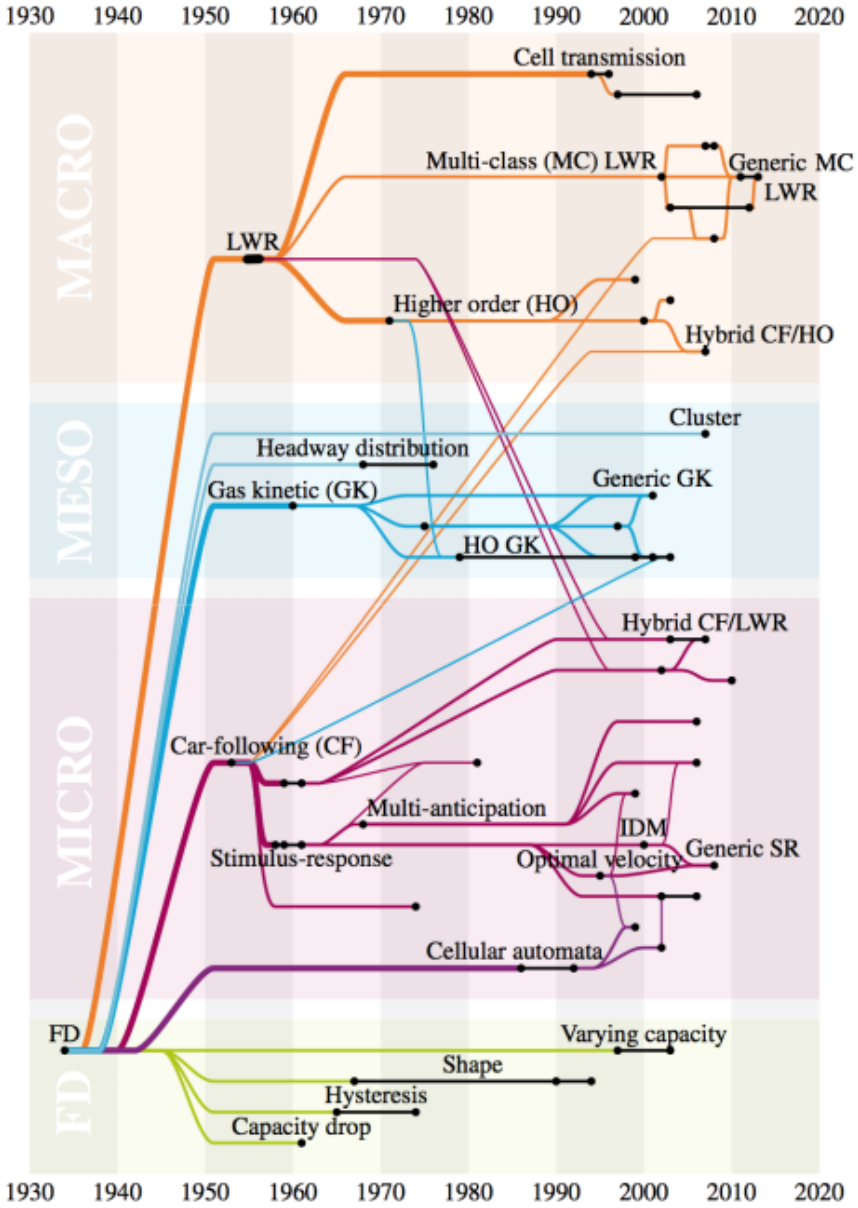


Figure 2.1: Genealogy of traffic flow models. Retrieved from [1].

As shown in Figure 2.1, there are several types of simulation models descending from the Fundamental Diagram(FD) of Greenshields [13]: macroscopic, mesoscopic, and microscopic models. The macroscopic model defines a simulation on the complete road flow using averaged quantities such as density and others. The microscopic model performs simulations treating each vehicle as an individual agent. Finally, the mesoscopic model is a combination of macroscopic and microscopic simulations.

The majority of traffic simulators use the microscopic approach due to detailed aspects inside the simulation and replicating driver behavior. The following section presents this model.

2.1.1 Microscopic Traffic Flow Models

The microscopic model simulates objects (vehicles) as an individual agent of the simulation. This implies calculating the actions (position, acceleration, speed, lane changes, and others) of the vehicle regarding the surrounded traffic at each simulation step. This model resulted in the development of car-following and lane-changing models.

2.1.2 Car-following Model

The idea of this model is that microscopic simulations describe the actions of each vehicle in function on the position and speed of vehicles nearby [14].

The first car-following model was introduced by Pipes in 1953 [15], and it was a safe-distance model. In this model, vehicles adapt their velocity corresponding to the leader vehicle to maintain a safe distance to other agents of simulation [1]. A formal definition on the classical car-following model are shown in Eq.2.1, where $v_i(t)$ represents the velocity of a i_{th} vehicle in time t, $v_{i-1}(t)$ is the velocity of the vehicle preceding the previous one, and τ is a relaxation on some time scale, representing how the driver achieves the desired velocity [14].

$$\frac{dv_i(t)}{dt} = \frac{v_{i-1}(t) - v_i(t)}{\tau} \tag{2.1}$$

The car-following model is represented in Figure 2.2, the $n-1$ object is the leading vehicle, the n vehicle is following it, and the variables l_n and l_{n-1} represent the length of said vehicles. The value of x_n and x_{n-1} represents the position of both cars. The value of d represents a safe distance, which means that the vehicle will not exceed that distance when close to the leader. Finally, T_{v_n} represents the safe stopping distance, with T representing the time and v_n the speed of the vehicle.

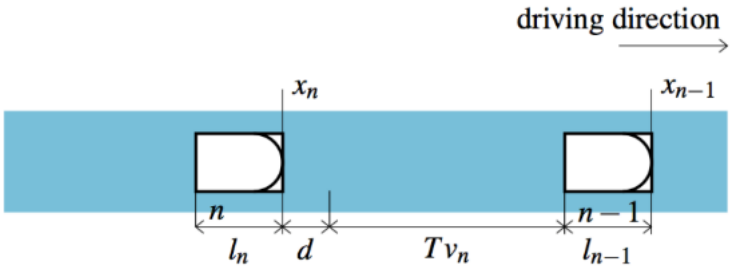


Figure 2.2: Pipes safe distance model. Retrieved from [1].

In later years, car-following evolved into several models such as the optimal velocity model [16] [17], cellular automaton [18][19], and even modelling of human behaviour [20]. Further historical analysis can be found in Brackstone and McDonald [21].

2.1.3 Lane Changing Model

The lane-changing model defines a decision-making model where a vehicle in the simulation contemplates several factors to change lanes in a multi-lane scenario. Several factors can

be critical such as traffic signals or traffic jam situations. Gipps presented the first model in 1986 [22]. Table 2.1 shows the performed steps towards a successful lane-change in this model.

According to Krauss [14], the topic of lane changes has reduced literature compared to car-following models. Sparmann [23] performed a two-lane-freeway analysis, Leutzbach and Busch [24] performed another analysis in three-lane-freeways, and Chowdbury et al. and Latour performed rules in cellular automaton [25][26].

Table 2.1: Decision process for changing lanes

Name	**Description**
Selection of lanes	The target lane is the lane where the driver intends to move. This lane could change due to the impossibility to move into the said lane.
Feasibility of changing lanes	To be able to change lanes, two conditions must be satisfied: the target lane must be one of the available to the vehicle, and said lane must be denuded of obstruction of other vehicles.
Driver behavior close to the intended turn	Ask whether the vehicle is close to the intended turn. If the answer is yes, then perform the change to that lane.
The urgency of changing lanes	As the vehicle approaches the intended lane, the urgency to turn increases.
Transit lanes and vehicles	Transit lanes are dedicated to the use of public transport and must be contemplated depending on the region simulated.
The entry of nontransit vehicles into transit lane	Ordinary vehicles could transit lanes in the presence of an obstruction.
The departure of nontransit vehicles from a transit lane	The model must contain the logic to ensure the ordinary vehicle leaves the transit lane as the blocking ends.
Driver behavior in the middle distance	If the driver has an intended turn and is near there, not all lane changes should be acceptable.
Relative advantages of present and target lanes	If the previous steps have no helped make a decision. The driver should evaluate the advantages of changing to a target or staying in the present lane.
Effect of heavy vehicles	Heavy vehicles tend to have a lower speed than ordinary ones so that vehicles can change lane in the presence of a heavy vehicle preceding them.
Effect of preceding vehicle	A vehicle could change lane if it analyzes that the preceding vehicle is slow and in the target, the lane could gain a significant speed.
Safety	The remaining question before changing the lane is whether we can safely perform it. If so, the model performs a lane change.

2.2 SUMO

Simulation of Urban Mobility (SUMO) [27] [28] [29] is a continuous and multi-modal traffic simulator that handles large network scenarios. It was developed by the employees at the German Aerospace Center starting in 2001. Figure 2.3 depicts the Graphic User Interface (GUI) of SUMO.

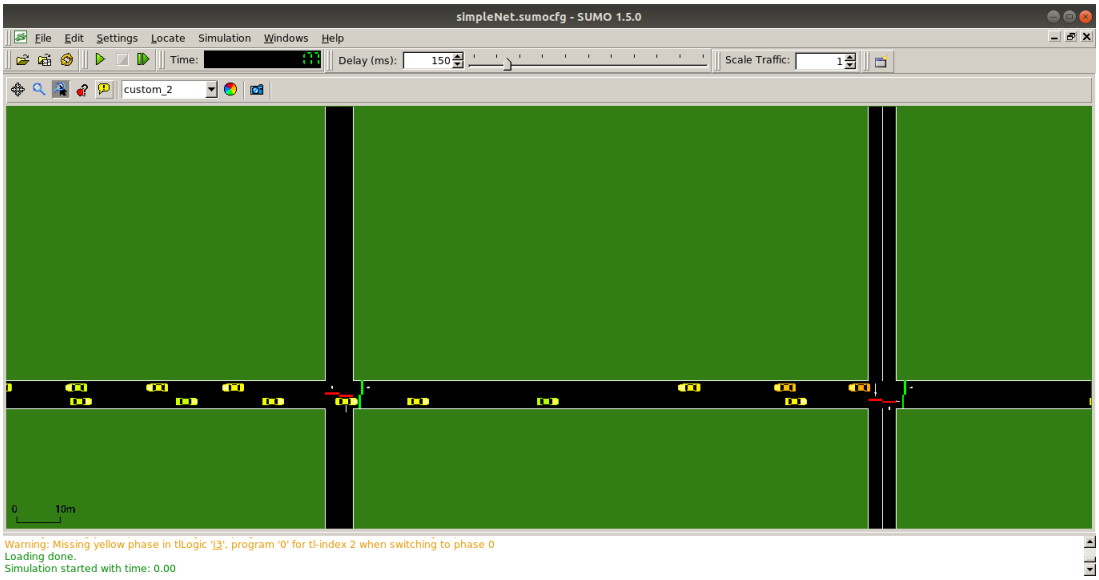


Figure 2.3: SUMO Graphic User Interface

The SUMO traffic simulation tool has many applications that help prepare the necessary elements to perform a simulation. The main elements to perform a SUMO simulation present two essential parts: representation of road networks and traffic demand [8]. In the following subsections, we present a description and examples of both representations.

2.2.1 Network Building

The network representation in SUMO is a directed graph. The nodes represent a point in the network using a coordinate within the map. The edges describe the union between two nodes, which characterizes a unidirectional street in the map. Edges need to have a starting and final node so that vehicles will know the street direction, and each edge can contain several lanes that run in parallel. Each lane contains constant values that represent the width, speed, and other features that limit the behavior of the vehicle transiting that lane. A network representation in SUMO is depicted in Figure 2.4.

A file (*.net.xml) embodies each network representation. This file contains all the elements mentioned previously and others that allow the correct interpretation of the entire network. Users can construct these files using tools such as NETEDIT and NETCONVERT applications. NETCONVERT is a command-line tool that allows the importation of networks from different sources such as OpenStreetMap (OSM) or other simulators, e.g., MATSim [30]. NETEDIT is an editor used to generate, examine, and update network files and allow manual manipulation of converted networks [27].

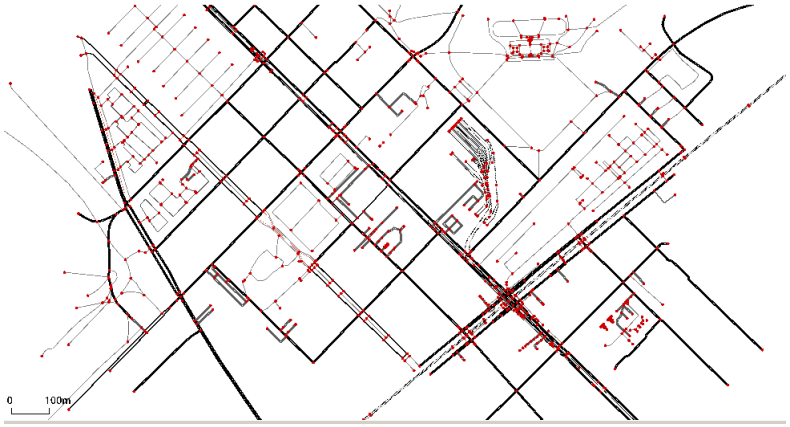


Figure 2.4: Network representation in SUMO

Network files contain detailed information regarding the representation of a graph encoded as an XML file, and the main contents include cartographic projection, edges, junctions, and connections. These files contemplate other elements, such as traffic lights and roundabouts. Figure 2.5 depicts the elements of a network file. The description of each element in the file is detailed below.

```
<location netOffset="500.00,200.00" convBoundary="0.00,-11.35,1000.00,400.00"
origBoundary="-10000000000.00,-10000000000.00,10000000000.00,10000000000.00" projParameter="!"/>

   <edge id=":gneJ0_0" function="internal">
      <lane id=":gneJ0_0_0" index="0" speed="12.08" length="13.47" shape="208.32,201.60 204.68,201.89 201.42,202.76 198.54,204.20 196.04,206.22"/>
   </edge>

   <junction id="gneJ3" type="priority" x="0.00" y="0.00" incLanes="gneE3_0 gneE10_0" intLanes=":gneJ3_0_0 :gneJ3_1_0" shape="-3.20,7.73 3.20,7.73
7.73,3.20 4.39,0.29 1.66,-1.15 -0.47,-1.13 -1.99,0.36 -2.90,3.31">
      <request index="0" response="00" foes="00" cont="0"/>
      <request index="1" response="00" foes="00" cont="0"/>
   </junction>

   <connection from="gneE0" to="gneE7" fromLane="0" toLane="0" via=":gneJ1_6_0" dir="r" state="M"/>
```

Figure 2.5: Network file representation

Coordinates

Network files use Cartesian projection to represent the spatial distribution. The leftmost node is at x=0 and the node at the bottom is at y=0. The label location represents such projections and the characteristics are described in Table 2.2.

Edges and Lanes

As stated before, an edge is a union between two nodes (junctions), and each edge can possess several lanes that are parallel unidirectional representations of streets. The characteristics of an edge is shown in Table 2.3 and a lane is represented in Table 2.4.

Table 2.2: Coordinate parameters in a network file

Name	Description
netOffset	The offset for moving the network to (0,0).
convBoundary	The boundary of the current network.
origBoundary	The boundary of the original network before projection.
projParameter	Information on how the network was projected.

Table 2.3: Edge parameters in a network file

Name	Description
id	Id of the edge.
from	Id of the node it starts at.
to	Id of the node it ends at.
priority	How important the edge is.
function	Edge purpose.

Junctions

The junctions can be seen as nodes in the directed graph and represent intersections between lanes. Table 2.5 presents the characteristics of them.

The junctions can posses several requests. The requests describe, for each link, the streams that have higher priorities. Table 2.6 presents characteristics of the requests.

Connections

The connections, or links, describe which lane the vehicle can reach from an oncoming lane, and it shows the first way to use it after passing the intersection. Table 2.7 gives the characteristics of connections.

2.2.2 Demand Modelling

After the network generation, we need to implement traffic demand. It possesses vehicles and their respective trip information. In SUMO, there are several ways of representing traffic demand, either as trips, flows, or routes [27]. This work exposes the representation in routes. We can declare several types of vehicles with different characteristics such as size, behavior, and others. The critical part of this file is creating each vehicle that will appear in the simulation. Each of them will possess a type, departure time, and a predefined route, represented as a set of edges that will travel during the simulation. A file that embodies this traffic demand is known as a route file (*.rou.xml).

To generate a route file, we can use a tool provided by SUMO called randomTrips. It is a python script that uses a network file and generates a series of routes in a file according to

Table 2.4: Lane parameters in a network file

Name	Description
id	Id of the lane.
index	A running number that starts at zero.
speed	Speed limit in the lane (m/s).
length	Distance of the lane (m).
shape	The contour representation.

Table 2.5: Junction (intersection) parameters in a network file

Name	Description
id	Id of the intersection.
x	The coordinate of the intersection in the x axis.
y	The coordinate of the intersection in the y axis.
incLanes	The set of lanes that finish in the intersection.
intLanes	A list of the lanes in the junction.
shape	Describes the limits of the junction.

parameters provided by the user. These routes are usually unbalanced, which means that vehicles will possess randomized characteristics. A different approach implies using Origin and Destination (O-D) matrices [31] to create demand with the tool OD2TRIPS provided by SUMO. Another essential tool called DUAROUTER provides the basics to understand the demand traffic and transformation of a trip to route files. Besides the mentioned tools, there are other essential tools to generate traffic scenarios such as ACTIVITYGEN, Flowrouter, DFROUTER, and JTRROUTER [27].

A route file usually contains detailed information on the traffic demand. Figure 2.6 depicts a representation of a route file. Detailed information on the representation is explained below.

```
<routes>
  <vType id="type1" accel="0.8" decel="4.5" sigma="0.5" length="5" maxSpeed="70"/>
  <route id="route0" color="1,1,0" edges="EW0 EW1 EW2 EW3"/>
  <vehicle id="0" type="type1" route="route0" depart="100" color="1,0,0"/>
</routes>
```

Figure 2.6: Route file representation.

Vehicle type

The first step defines one or several types of vehicles (vType) in the simulation. Each type declares several variables used to create the vehicle shape and behavior through the entire simulation. Some of these characteristics are shown in Table 2.8.

Table 2.6: Request parameters in a network file

Name	Description
index	The index of the connection.
response	Bitstring representing whether it prohibits the un-decelerated passing of the intesection.
foes	Bitstring describing conflicts of other connections with the actual one.
cont	Whether a vehicle may pass the first stop line to wait until there are no other vehicles with higher priority.

Table 2.7: Connection parameters in a network file

Name	Description
from	Id of the incoming edge.
to	Id of the outgoing edge.
fromLane	Id of the incoming lane.
toLane	Id of the outgoing lane.
via	The first lane to use after passing the connection.
dir	Direction of connection.
state	State of the connection.

Route

The route defines a set of edges and other characteristics that will serve as a path for a vehicle within the simulation. All the straight edges in the set must connect them to avoid errors in the simulation. Table 2.9 presents the characteristics of the routes.

Vehicle

The vehicle defines the simulated object that will move through the entire simulation in the desired network. Table 2.10 shows the essential characteristics of the vehicles.

2.2.3 Car-following and Lane-changing Model in SUMO

The current car-following model used in SUMO is a Gipps model extension [14] [32]. In each time step, the speed of a vehicle adapts to the speed of the leading vehicle in a way that avoids collisions in the next time step. This velocity is called vsafe, and Equation 2.2

Table 2.8: Vehicle Types in a route file

Name	Description
id	Name of the vehicle type.
accel	Acceleration ability of this vehicle (m/s^2).
decel	Deceleration ability of this vehicle (m/s^2).
sigma	The driver imperfection for car-following model
length	The vehicle length (m).
maxSpeed	Maximum speed allowed for the vehicle.

Table 2.9: Route representation in a route file

Name	Description
id	Name of the route.
edges	List of edge ids that the vehicle will drive along.
color	The color of the route.

states this computation.

$$v_{safe}(t) = v_l(t) + \frac{d(t) - v_l(t)\tau}{\frac{\bar{v}}{b(\bar{v})} + \tau} \tag{2.2}$$

Where $v_l(t)$ is the velocity of the front vehicle, $d(t)$ is the length to the front vehicle, τ is the response rate of the driver, and b is the deceleration function [28]. The desired velocity of the driver is the minimum between the safe velocity, the maximum velocity, and the actual velocity plus the maximum acceleration as shown in Equation 2.3.

$$v_{des}(t) = min[v_{safe}(t), v(t) + a, v_{max}] \tag{2.3}$$

A random "human error" may also be added. However, that is not the project goal. More detailed information on the car-following model is presented by Krajzewicz et al. [28]. SUMO also implements other car-following models, and the documentation presents the available list.

In the lane changing model, SUMO implements a 4-layered hierarchy model [33] to determine the vehicle behavior in each time step. The scenarios taken in the count for the desired lane changing are:

- Evaluating subsequent lanes.
- Determining urgency.
- Speed adjustment to support lane changing.
- Preventing deadlock.

Table 2.10: Vehicle representation in a route file

Name	Description
id	Name of the vehicle.
type	Id of the type to use for this vehicle.
route	Id of the route that will circulate the vehicle, edges can be explicitly declared.
color	The color of the vehicle.
depart	The time step at which the vehicle enters the network.

The model computes previous scenarios to perform a lane change. First, it considers the right lane. If no change is possible, then a left lane is considered, and this model showed some positive outcomes in the performance compared to other approaches.

2.2.4 TraCI

Traffic Control Interface (TraCI) [2] provides access to an executing traffic flow simulation in SUMO, permitting the user to recover variables of simulated agents and manage their actions. TraCI employs a design based on TCP/IP that grants control over SUMO. The architecture matches a client/server topology. Thus, SUMO operates as a server that starts with command-line operations using a TraCI script (usually in Python/C++) and manipulates objects within the simulation in real-time. TraCI supports multiple clients and performs the desired actions in the clients until it calls the simulation step function. Figure 2.7 represents the TraCI architecture.

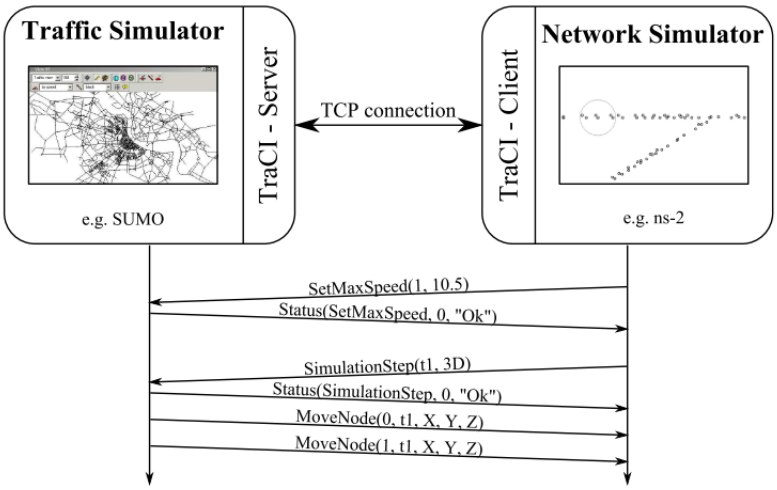


Figure 2.7: TraCI architecture. Retrieved from [2].

In the protocol shown in Figure 2.7, the client application sends commands to SUMO to control the simulation execution, an individual vehicle behavior, or retrieve values from simulation objects. SUMO answers with a status-response to each command and some additional values depending on the request.

The preferred language to write scripts using TraCI is Python as this posses complete documentation, supports all the TraCI commands, and the community test this library daily. Other languages support TraCI libraries, such as C++ or Matlab, but they do not possess complete support. To achieve a connection between SUMO and TraCI using Python, we have to import TraCI in the script. The SUMO_HOME/tools directory must be in the python load path. Figure 2.8 shows the way of achieving this.

```
if 'SUMO_HOME' in os.environ:
    tools = os.path.join(os.environ['SUMO_HOME'], 'tools')
    sys.path.append(tools)
else:
    sys.exit("please declare environment variable 'SUMO_HOME'")
```

Figure 2.8: Import TraCI in a Python script.

After importing TraCI to the Python load path, we start our simulation, and connect to it using the script created. A basic Python script that uses TraCI to perform 1000 simulation steps is depicted in Figure 2.9.

```
import traci

sumoBinary = "/path/to/sumo"
sumoCmd = [sumoBinary, "-c", "yourConfiguration.sumocfg"]

import traci
traci.start(sumoCmd)
step = 0
while step < 1000:
   traci.simulationStep()

   step+=1

traci.close()
```

Figure 2.9: Basic TraCI script.

After connecting to the simulation, several commands can be executed and then perform a simulation step until required. Finally, the simulation must be closed.

2.3 MPI

Message Passing Interface (MPI) is a standard specification for message-passing calls within processes defined by the MPI Forum in 1994 [34] [35]. The specification includes point-to-point communication and global operations [36]. The message-passing model has proven to be efficient for parallel systems with distributed memory and distributed systems. It is also widely used in both homogeneous and heterogeneous systems due to the architecture of the implementations. Several implementations from the MPI standard have been developed, such as OpenMPI[37], mpich[38] and many others. To send information in MPI, we have two types of data that can be transmitted. One is information regarding the execution of the programs, and the other one is actual data to be stored in the receiving side of the

communication. Data transfer could take significant time, depending on the size, and it performs in either point-to-point or collective communications. Point-to-point is the communication between two processes that exchange information and collective communication between one process and all the other processes.

2.3.1 Groups, Contexts, and Communicators

A group is a collection of processes that can communicate, point-to-point and collectively, between them. In terms of MPI, groups present ordered sets of process identifiers. Each process is assigned a number between 0 and the number of tasks. A context that isolates each point-to-point communication performs these communications. Each message sent within a context can be received only in that context so that messages are sent and received by the desired processes.

A communicator is an MPI object that wraps the idea of groups and contexts. This communicator provides communication services between the processes. MPI defines a communicator called MPI_COMM_WORLD, a group with a defined process and a unique context [39].

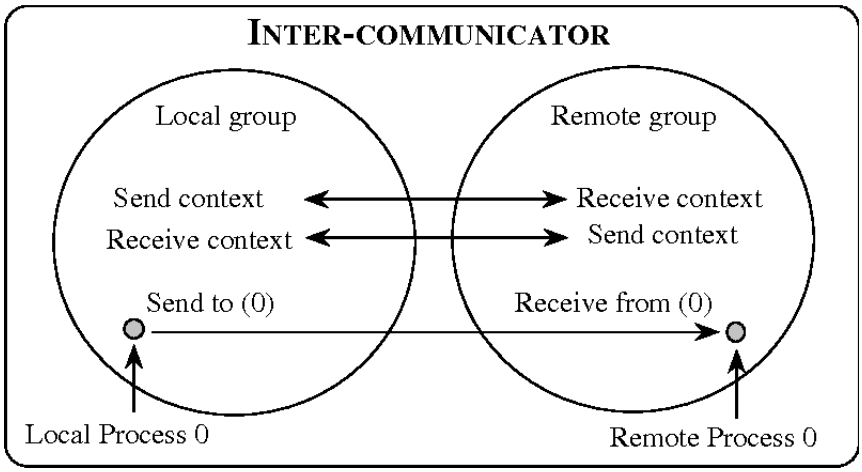


Figure 2.10: Group, context and Communicator example in MPI. Retrieved from [3].

2.3.2 Point-to-Point Communication

The basic operations of point-to-point communication are the send-receive routines. They send a message between a pair of processes with a source, destiny, and tag explicitly defined. The send (MPI_SEND) and receive (MPI_RECV) are performed in a blocking form. All processes are blocked until a return is received, in both sender and receiver. There are four blocking primitives in MPI presented in Table 2.11.

MPI also provides a non-blocking communication where a process can continue executing while a buffer is either sending (MPI_ISEND) or receiving (MPI_IRECV) a message. Some functions allow the test and wait of messages transmitted in non-blocking communication [39]. However, a barrier can also halt all the processes execution until all communications complete.

Table 2.11: Blocking send in MPI

Primitive	Description
Synchronous	Completes once the sender receives the acknowledgment of the receiver.
Buffered	The send is blocked until the message copies into a buffer in a message buffer scenario.
Standard	The sender is blocked until the send buffer frees, and the receiver is blocked until the message is in the received buffer.
Ready	The message is sent, and if there is no receiver waiting, then it is dropped.

2.3.3 Collective Communication

In collective communications, all processes within a group are involved in the operation. This can be helpful in terms of synchronizing an application. Table 2.12 shows some of the collective operations.

Table 2.12: Collective operations in MPI

Operation	Description
MPI_BARRIER	Synchronization of all the processes in the group.
MPI_BCAST	Send a broadcast message to all processes from a root.
MPI_GATHER	All processes send data to a root that stores them in order.
MPI_SCATTER	A root sends data to all processes in rank order.
MPI_REDUCE	Uses data from all processes, perform an arithmetic or logical operation, and sends it to a root.

2.3.4 MPI for Python

MPI for Python (mpi4py) [40] is a library that allows user applications to exploit the idea of multiple processors using MPI standards in a Python script. This package provides a

designed interface that translates MPI syntax from C++ to Python. The script written in Python calls a wrapper that contains an extension module written in C.

This library implements several classes representing the MPI standards, such as communicators, point-to-point, and collective operations. The primary communicator is known as WORLD. We also have a Get_rank() function that allows creating a group within the communicator and communication techniques like send() and receive(). A performance analysis performed in this package showed similar behavior in time execution than the C implementation [40]. The overhead imposed by MPI for Python is relatively smaller than those presented in the literature of interpreted versus compiled languages using C.

Chapter 3

State of the Art

This chapter presents preliminary research regarding the topics presented in this project. Each subsection provides distributed traffic simulation approaches from a current perspective. The organization of these subsections is presented next. Subsection 3.1 presents some partitioning algorithms for traffic simulations. Subsection 3.2 presents some approaches for distributed traffic simulations.

3.1 Partitioning Algorithms

Partitioning algorithms propose methods to effectively divide a required network within a traffic model and use it in parallel/distributed simulations to increase performance. Some of the partitioning algorithms in the literature are described below.

3.1.1 Graph Partitioning

As explained in Chapter 2, a traffic simulation network can simulate a graph problem where the junctions represent the nodes and the edges bonds them to form a street within this network. For this purpose, we can use several partitioning algorithms in this context.

Graph partitioning algorithms have proved to be NP-Hard problems. Solutions for these problems tend to use heuristics and approximations. The main algorithms implement multi-level and spectral methods. Some of these methods were introduced in the early years [41] and implemented in packages such as METIS [42]. Some graph partitioning algorithms have been used in traffic simulation networks to improve the obtained balance in big scenarios.

GrapH [43] is a distributed graph processing system that relies upon dynamic vertex traffic and costs to perform partitions in a network. This method implements a fast single-pass algorithm, named H-load, and a method for online traffic prediction that treats each vertex as an individual learner to predict vertex traffic patterns. The results showed that GrapH could outperform other algorithms in terms of communication costs and end-to-end latency.

Ahmed and Hoque [4] presented a partitioning algorithm based on a list of parameters such as node weight, link lengths, number of lanes, link density, and link priority. They

used METIS to partition the network and generate a node and link weighted graph. Figure 3.1 depicts the approach presented by them. Although it looks promising, this algorithm has not been tested in the literature, leading to a gap regarding its performance.

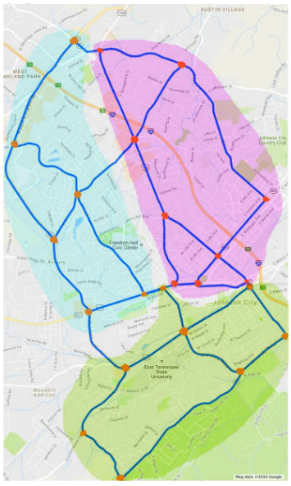


Figure 3.1: Graph Partitioning. Retrieved from [4].

Graph partitioning can be seen as an optimal solution to achieve an efficient network partition for traffic simulations. However, these approaches tend to present a high level of complexity to simple road networks, and the main topic of this project is not partitioning. For this purpose, a space partitioning algorithm is better suited to performance analysis of the implemented system.

3.1.2 Space Partitioning

Space partitioning is the action of dividing a determined space into non-overlapping regions. These algorithms are generally hierarchical, where an area is divided into regions, and the regions divide recursively. This behavior forms space-partitioning trees. The algorithms usually are divided into uniform and non-uniform approaches. The first one employs static and regular shapes, such as rectangles, to perform partitions. The second one presents areas that are irregular and not static that generally presents a hierarchical approach.

SPartSim [5] is a multi-level region growing road network partitioning based on a non-uniform hierarchical approach. In this algorithm, they used an initial vertex as a starting point and started growing the region from that vertex. Once all regions had reached the boundaries of the network or contacted other growing regions, a balancing algorithm start to trade segments lengths between partitions to accomplish a more balanced network partitioning. The SPartSim result is depicted in Figure 3.2. The resulting partition showed improvements regarding the number of road cuts and load-balancing than traditional solutions(i.e., Simple quad-tree partitioning [44]). However, it showed a higher execution time required to perform the partition.

The partition algorithm used in this project is presented by Acosta et al. [6]. This is a uniform space algorithm where the network is partitioned in even slices depending on the number of nodes required for the partition. Figure 3.3 presents an example of

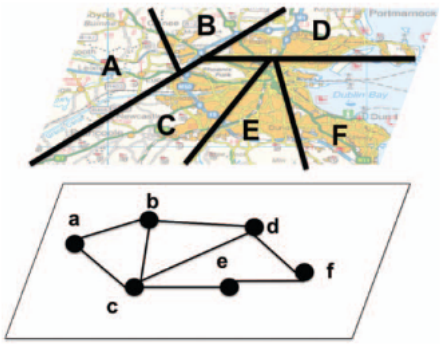


Figure 3.2: SPartSim Partitioning. Retrieved from [5].

this approach. The authors use NETCONVERT to perform the partition and provide an algorithm to patch the differences between the original edges and the ones generated in the partition. This algorithm uses SUMO tools to achieve the partitions, and the implementation is detailed. However, the described approach has proven to be inefficient in unbalanced networks [6].

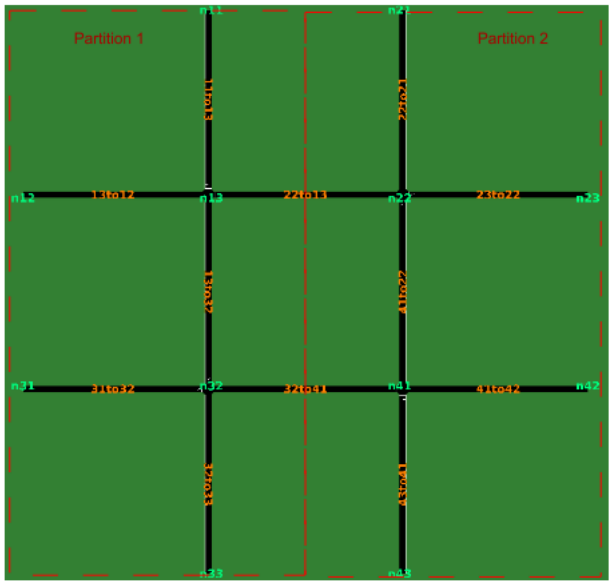


Figure 3.3: Uniform Space Partitioning. Retrieved from [6].

For this project, we are using a balanced network to perform the simulations, and a uniform space partition algorithm is sufficient for the scenario. In the future, we will employ a graph algorithm to work with more unbalanced networks, such as big-city scenarios.

3.2 Distributed Traffic Simulations

The appearance of different traffic simulators has led to a growing research area to improve the performance of simulations. For this purpose, the literature presents some approaches

using High-Performance Computing techniques, such as parallel or distributed implementations using existing applications and creating new ones (i.e., GeoSparkSim [45]).

In the case of SUMO, the absence of an official distributed implementation and the lack of literature motivated this project. There are two implementations of distributing a SUMO simulation. The strategies used are described below.

The first approach is called dSUMO [7] is a distributed version of SUMO, using different cores on a machine or different machines. In this implementation, each node runs a SUMO instance and a partition of its network. The communication performed in this system use sockets. Figure 3.4 depicts the architecture of dSUMO. The idea is to have a container that runs within a machine and runs one or several SUMO instances using a handler and runner to manipulate the simulation and a client-server interface to communicate with other containers. There is only one server and handler, but we can have several clients and runners depending on the number of partitions we are using.

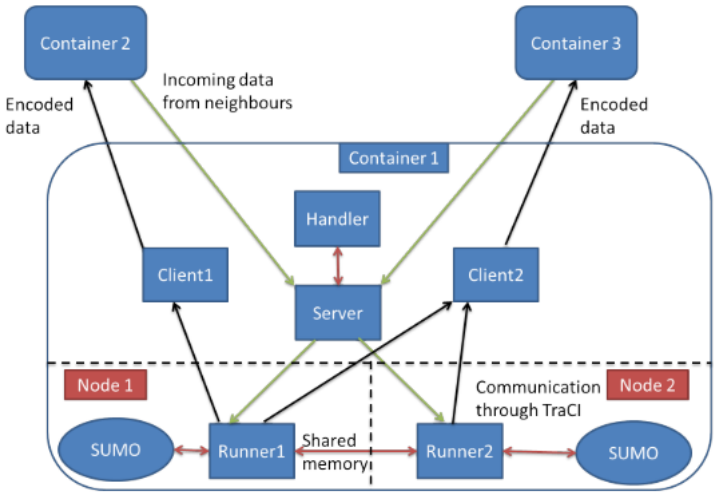


Figure 3.4: dSUMO architecture. Retrieved from [7].

The handler serves as an operator interface that provides full functionality obtained using SUMO. The runner uses TraCI to interact with SUMO. It creates links with other nodes using a shared-memory approach and serves as the main component to manipulate the simulation. The client sends the required information to all the neighbor containers, and the server receives incoming messages from clients.

Several messages are created to send among containers, such as the vehicle message that sends a message of vehicles crossing the borders. The main idea of synchronization lies in back-controlling a car. When a vehicle leaves one partition to enter another, it will send back position and speed to the leaving partition for a short period. This idea does not affect the car-following model and the general result in the simulation. The results showed accuracy up to 99%, and that dSUMO had better simulation time than the centralized version in terms of SUMO execution time. However, synchronization seems to add an unnecessary overhead in this implementation.

Acosta et al. [6] performed an analysis in the network partitioning and border edges management. They used the network partition method explained in the previous subsection and three ideas to implement the border edge management. One included replicating a

model. The other performs braking when the leading car stops according to the minimum safe gap. The last one uses a virtual traffic light to control the behavior of incoming vehicles. Their results showed that the traffic light approach presented minimum errors. However, lights did not contemplate the usage of yellow lights, and this introduces a limitation.

Further research by Arroyo et al. [46] implemented an idea similar to Acosta et al. [6] and a synchronization method using a master-slave scheme. They implemented a partition using a "dead-end" approach that included one of the nodes in the partition to have a dead-end in the edge that will be shared, and the vehicles in both partitions could co-exist. Vehicles in the master partition limited the behavior of the exact vehicle in the slave partition, and a new vehicle inserted in the slave partition will also be inserted in the master. They implemented shared-memory within the TraCI packages and resulted in a reduced time of 60% compared to the traditional TCP/IP approach. The results showed error values near 0% with an increased number of outliers for congestion scenarios.

Distributed simulations have been achieved using other simulators but with the same partition approach. Distributed Urban Traffic Simulator (DUTS) [47] is a distributed version of the simulator Java Urban Traffic Simulator (JUTS). It considers multiple core computers in a cluster environment and performs analysis using several nodes and threads to analyze the performance of the implementation. The system uses a shared-memory architecture to perform communication between cores and message passing between nodes. It uses a particular node called a control process that synchronizes and controls the entire simulation. The spatial partition is performed by dividing the marked lanes in half. The approach takes advantage of the multi-core computers to implement a parallel/distributed system and compare it to a purely distributed system. The results show that the multi-core approach reaches a 52% speedup compared to a distributed implementation.

In this project, some ideas for implementing the network partition were based on the work of Acosta et al. [6]. The border management is similar to the back-controlling of a car from dSUMO [7] as this showed reduced errors in a distributed simulation.

Chapter 4

Methodology

This chapter presents the performed procedure to accomplish the required objectives. First of all, we present the problem-solving scheme in chronological order. Then, the proposed model is introduced, which includes the principal aspects for implementing the distributed simulation. Ultimately, we propose the scenarios and techniques to interpret our results.

4.1 Phases of Problem Solving

The workflow diagram used to perform this project is depicted in Figure 4.1.

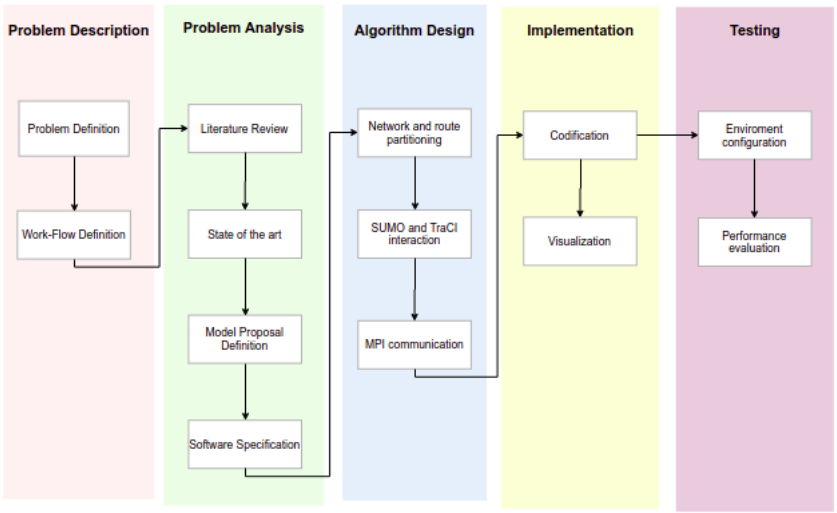


Figure 4.1: Phases of Problem Solving

4.1.1 Description of the Problem

This phase describes the project development pillar as it introduces the central issue, its latent obstacles, and how to overcome them. For this purpose, we introduced the concepts of traffic simulation, SUMO, and MPI. Later, we defined the potential problems

to address in detecting vehicles and communication between nodes. Finally, we present a model for implementing the distributed traffic simulation problem. To achieve this, Figure 4.1 presents a workflow to produce the model scheme. For this purpose, we chose to implement a partitioning algorithm and an application from scratch that efficiently controls the simulations and runs in a distributed system. Chapters 1 and 4 condense the actions displayed in this stage.

4.1.2 Analysis of the Problem

This phase presents some background and essential information required to understand this project. Chapter 2 systematically exhibits this knowledge, where the traffic flow simulation is analyzed. Then, we present an introduction to the SUMO simulator, and finally, the basics of MPI and its inference in distributed systems are detailed. This data helped us to present a concise state of the art. Regarding this information, we established a contemporary model proposal, where we propose the partition techniques, simulation characteristics, and communication interface. Finally, the software employed is specified: SUMO to perform the simulation and Python for control interface and communication setting. Chapters 2 and 4 present the learning specified in this stage.

4.1.3 Algorithm Design

During this stage, we produced the described approach in the model scheme. First of all, we developed an implementation of a network and route partition algorithm. A peer-to-peer network interface was then created using MPI to communicate partitions between different distributed system nodes. Section 4.2 explains these algorithms.

4.1.4 Implementation

This phase describes software requirements used for each of the steps during the development of the project. For the simulation, the SUMO simulation package was used due to its microscopic behavior and relevance. To control the behavior of the simulation in real-time, we used the TraCI library in Python because of the documentation and support provided in this language. Finally, an MPI library in Python (mpi4py) was selected as we are using it as the principal language in the project.

4.1.5 Testing

This part concentrated on estimating and explaining the achievement of the implementation in a distributed traffic simulation. For that purpose, we perform several simulations using different vehicular densities to appreciate the algorithm behavior in a scenario. The simulations were performed in several computing nodes to compare performance between this implementation and the traditional approach. Finally, the results are evaluated regarding time requirements to compare the obtained performance. Section 4.3 details all these steps.

4.2 Model Proposal

This project proposes a model to implement a distributed traffic simulation using SUMO that will perform a single core simulation job in several computing nodes to increase the performance in the required computations during each time step of the simulation. These nodes need to communicate between them to allow vehicle transference in real-time. Several factors must be considered to achieve this, such as the partition of the network/route and the communication interface. In this sense, this model implements a communication based on MPI basics and some literature review principles. For this purpose, we performed several steps. First, a network and partitioning algorithm split the required files. Then, SUMO and TraCI established an interaction to control the borders in the partitions, and the communication protocol, using MPI routines, was defined. Finally, we implemented the algorithm using several libraries and Python.

4.2.1 Network Partitioning

Several partition algorithms can split networks into different portions that will contain different vehicles [5]. There are many partitioning algorithms based on space or graph partitioning. A space-based algorithm will be better suited for this problem as graph partitioning tends to be much more complex for simpler networks, as presented in this project.

The method of partitioning the network is similar to the one presented by Acosta et al. [6]. For example, we assume to have a network as presented in Figure 4.2.

Figure 4.2: Network example in SUMO

The uniform space algorithm used performs a non-disjoint separation of the network graph. For simplicity, we will refer to it as a partition. Figure 4.3 depicts the network partition required for two computing nodes. Two exact pieces divide the network, where each of the divided portions will belong to a different node in the system, and each node will be responsible for the network that lay on that side of the partition.

To preserve network topology, edges that are "cut" during the partition must remain intact in both partitions. The network in Figure 4.3 will be partitioned in two networks **A** and **B**. Each partition will be assigned to one of the two nodes. Figure 4.4 and 4.5 depict this behavior.

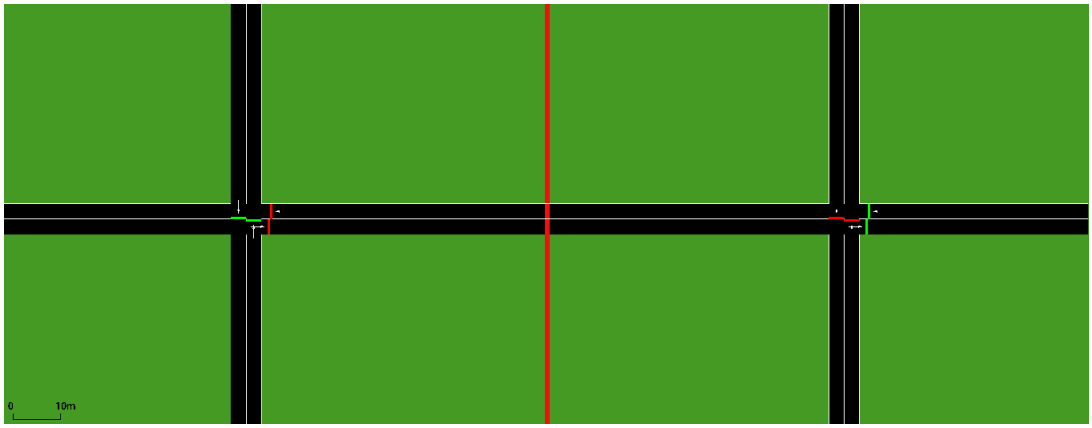


Figure 4.3: Network Partition for two nodes in SUMO. The red line represents the virtual border.

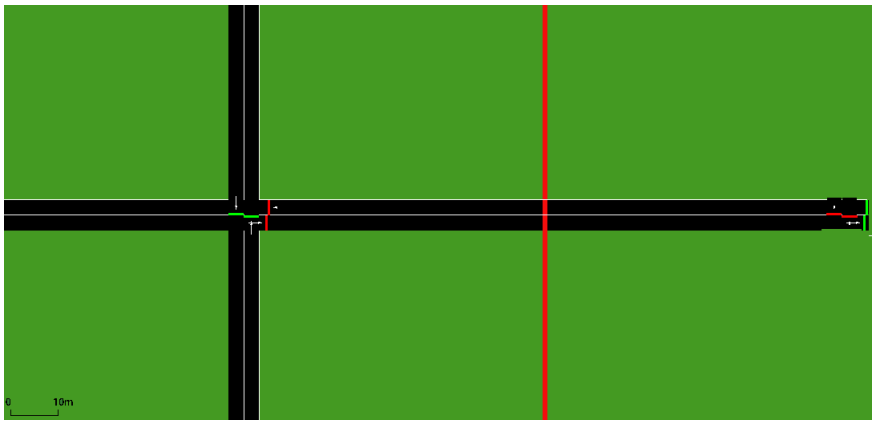


Figure 4.4: Network Partition for **A**

The spacial partition is performed using NETCONVERT and the option –keep-edges.in-boundary to explicitly declare a required boundary within the network limits. However, when using this application, the resulting boundaries can be affected due to edges converted into dead ends. To this problem, it was necessary to reevaluate the new position of the nodes within the new partitioned network and compare them with the original network to patch the difference between them.

As stated before, each node will receive a partitioned network and will be responsible for all the vehicles that start the simulation in that partition. Thus, a route partition regarding the new networks is also required. We need to assign the vehicles that will depart in partition **A** to a new route file and assigned to the node in charge of **A** and the same for vehicles in **B**.

4.2.2 Route Partitioning

As mentioned before, after the network partitions are created, we need to create the corresponding route files for each partition. This partition will take a vehicle that starts in a partition and put the corresponding routes of the vehicles in a new file. The sigma value

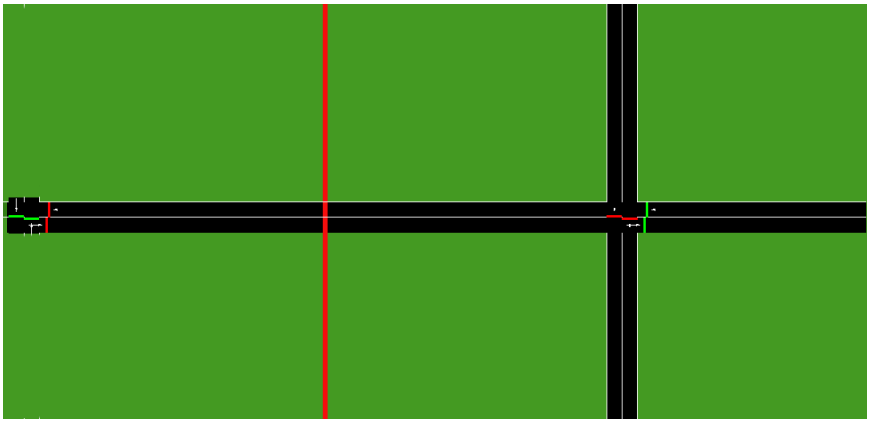


Figure 4.5: Network Partition for **B**

for the vehicle type is set to 0 as we want to minimize the human error introduced in the car-following model and have the same simulation output in each execution.

An example of this partition is shown in Figure 4.6, 4.7 and 4.8. We assume to have two partitions **A** and **B** and a vehicle depicted in Figure 4.6 that will start in **A** and end in **B**. The edge that is partitioned in the network is **gneE0**, this edge will be present in both partitions as explained before. The route partition for **A** is shown in Figure 4.7, the vehicle will conserve the main characteristics of the original route and will contain the edges until **gneE0** and the route partition in **B**, depicted in Figure 4.8, will contain the rest of the edges, including **gneE0**, to allow the inclusion of the vehicle when it leaves **A** and arrives in **B**.

```
<vType id="type1" sigma="0"/>
<vehicle id="8" type="type1" depart="8.00">
   <route edges="gneE3 gneE4 gneE0 gneE7 gneE15 gneE14"/>
</vehicle>
```

Figure 4.6: Route of a vehicle

```
<vType id="type1" sigma="0"/>
<vehicle id="8" type="type1" depart="8.00">
   <route edges="gneE3 gneE4 gneE0"/>
</vehicle>
```

Figure 4.7: Route of the vehicle in the departing partition

```
<vType id="type1" sigma="0"/>
<route id="8" edges="gneE0 gneE7 gneE15 gneE14"/>
```

Figure 4.8: Route of the vehicle in the receiving partition

Algorithm 1 presents the main idea of the implementation using this approach.

Algorithm 1: Route Partitioning Algorithm

```
Read network and route files;
for each network in partitions do
    Make a list with all the edges within the partition;
    for each route in routes do
        if Departing edge in network boundaries then
            Create a new vehicle with the same characteristics;
            Assign edges within the partition to the new vehicle;
        else
            create a new route with the edges outside the partition;
        end
    end
end
Write route partitions in new files;
```

The route partition algorithm is implemented in Python, using sumolib, a library that allows reading and feature extraction of SUMO files and xml.dom.minidom library to create a route file template and use it to write information in new files.

4.2.3 Border Management

TraCI is in control of this part of the implementation. At every time simulation step, the position of vehicles is compared to the boundaries of the partition to identify when they are leaving the partition. Then it will operate the leaving vehicle to retrieve all the information and prepare it to send into other partition. After everything is ready, it will send the vehicle to the other partition. Once a node receives a vehicle, it will be inserted in the simulation using the received information. We use some commands such as TraCI.add, TraCI.remove to insert and remove vehicles during the simulation.

To ensure accuracy in the system, the model for managing the border is similar to the one presented in dSUMO [7]. A relevant aspect of the simulation is the car-following model and its functionality. The main goal is that if a car leaves one partition to another, it will not disappear from the existing partition as this will be leading some other vehicles. If a leading vehicle suddenly disappears, the vehicles following it will start to have a different behavior as they do not have any information about the vehicle ahead. This problem is solved by making a vehicle (**V**) leaving partition **A** and being transferred to **B** to exist in both partitions. The node that manages partition **B** will send information of **V** back to the node managing **A**. **A** will upload **V** position and speed according to **B** information until **V** leaves the edge circulating before transference. After **V** leaves said edge, it will stop existing in **A** and will only exist in **B**. This process is known as back-controlling a vehicle. In this approach, other vehicles following **V** will act as the standard simulation.

After the TraCI.add function is applied, the insertion of a SUMO vehicle will be per-

formed in the next time step of the simulation. Thus, it is vital to calculate the position of the vehicle in the next time step. Equation 4.1 presents the calculation of the following position, where $P_{i+1}(v)$ is the position of the vehicle in the following time step, $P_i(v)$ is the current position, $s(v)$ is the actual speed of the vehicle and Δt is the time step (usually 1 second). Note that the operation sign implies either an addition or a subtraction. This is defined by the edge direction in which the vehicle is circulating.

$$P_{i+1}(v) = P_i(v) \pm s(v) * \Delta t \tag{4.1}$$

4.2.4 Communication Protocol

The communication implementation uses an MPI library for Python (mpi4py)[48]. Figure 4.9 depicts the behavior of the system where each node will run a SUMO instance that contains a partitioned version of the network and the corresponding route file. The simulation of each node connects to TraCI, which will perform the real-time operations in each simulation step. A simulation manager will then be in charge of starting the simulation and performing a simulation step when required. The border manager performs computations on each vehicle between the simulations to calculate when a vehicle leaves a partition and starts another and back-controls a car after vehicle transference. Finally, a vehicle manager is in charge of sending and receiving vehicles, and more information, to other nodes using MPI communication and manipulating them using TraCI. The vehicle manager will add and remove vehicles within the simulation.

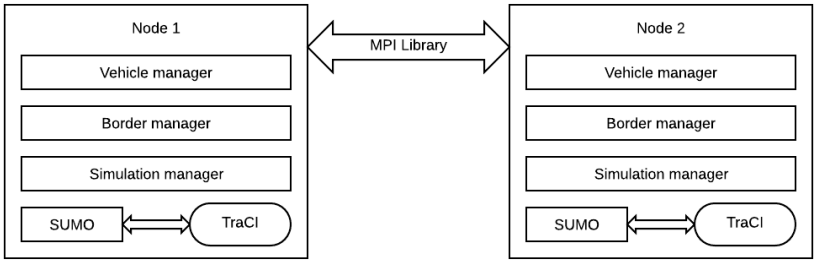


Figure 4.9: Architecture of Distributed SUMO

The MPI implementation uses peer-to-peer topology, non-blocking communication, and a barrier to guarantee thorough communication between nodes. This type of communication avoids blocking when several nodes try to communicate simultaneously. The barrier is placed after the communication part of the algorithm to wait until all nodes have communicated successfully.

4.2.5 Algorithm Implementation

Algorithm 2 presents a basic logic of the distributed simulation. As explained in previous sections, each node will start a SUMO instance with the partitioned network and route files. After the simulation start, we calculate the position for the next time step of the

simulation for each vehicle to find if some vehicle is leaving the partition. In that case, we send the vehicle using the MPI.isend function to perform non-blocking communication to the receiving partition. A false variable will be sent if there is no vehicle leaving the partition. Each time step, the receiving partition will be waiting with MPI.irecv to message the adjacent partitions. A barrier is placed after the synchronization process to guarantee thorough communication between the partitions. After the synchronization process, if the partition received a vehicle, it will be added using the TraCI.add command to the desired position. We need to check if some vehicles need to be back-controlled or removed, as explained in the border management. Finally, we perform a simulation step in all the partitions simultaneously. Figure 4.10 depicts a graphic representation of the algorithm.

Algorithm 2: Distributed SUMO algorithm

```
1  Initialize MPI.COMM_WORLD ;
2  for each node do
3      run a SUMO instance with partitioned networks and routes;
4      while Simulation is not over do
5          for v in vehicles that will leave the partition do
6              if v.position ± v.speed * timestep not in the partition then
7                  Send v to the destiny node.
8              end
9          end
10         Receive incoming messages;
11         MPI.barrier();
12         if Vehicle v is received then
13             TraCI.add(v);
14         end
15         Check for back-controlling and vehicles to be removed;
16         Perform a simulation step;
17     end
18     Finish the simulation and close SUMO;
19 end
```

4.3 Experimental Setup

The implementation of distributed SUMO using TraCI and the mpi4py library was tested using the Imbabura Cluster, established in Yachay Tech University in 2018. The cluster works with 35 workstations. Each workstation runs the operating system Ubuntu 18.04 and has Intel Xeon 3.6 GHz Quad cores with 32 Gigabytes of RAM. Figure 4.11 shows a

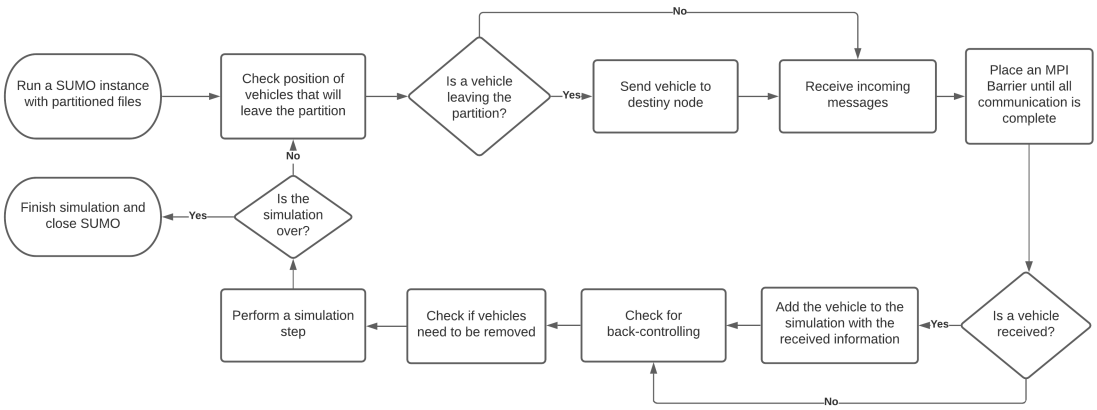


Figure 4.10: This workflow represents the behavior of one node executing the distributed implementation.

referential diagram of the cluster topology, and the representation is referential as we do not have root access to the cluster.

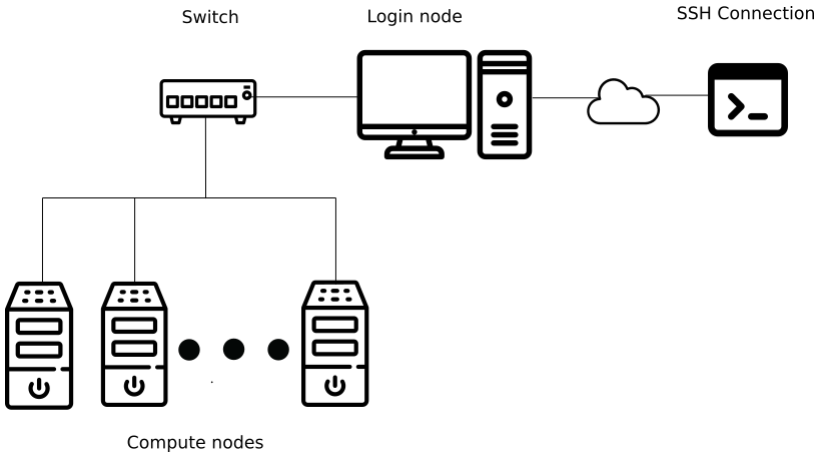


Figure 4.11: Imbabura Cluster Topology

The partition algorithm and distributed implementation were performed in Python 3.7.4, SUMO 1.6.0, and mpi4py 3.0.3 and the required libraries in SUMO source (TraCI, sumolib). Section 4.2 details the approaches employed for the algorithms.

The experimental setup for the distributed simulation used vehicular densities of 20, 40, 100, 500, 1000, 2000, 3000, 4000, 5000, 7500, and 10000 vehicles per simulation using 2,4 and 6 nodes. The network partitions are presented in Figure 4.12, Figure 4.13, and Figure 4.14 for two, four, and six nodes respectively.

4.4 Performance analysis

The analysis of the obtained results is performed in terms of time requirements of the execution in said simulations. This section describes the metrics required to test our

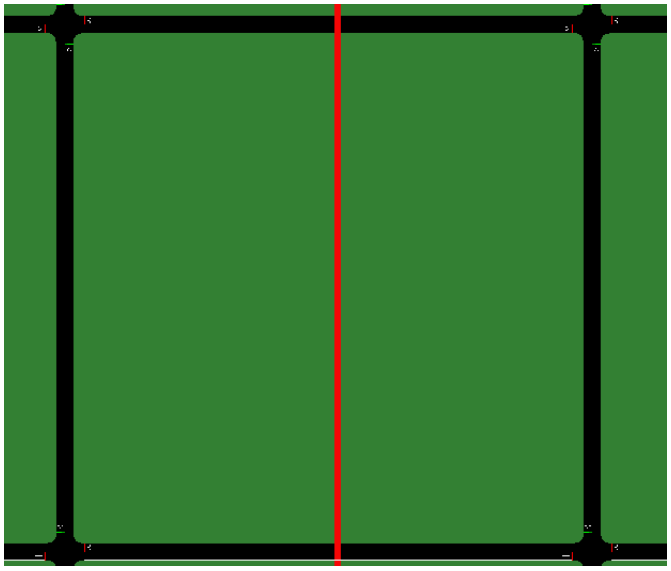


Figure 4.12: Network Partition for two nodes. The red line represents the partition border.

implementation.

- **Execution time:** It describes the time required to perform the entire, or parts, of the simulation. Equation 4.2 presents this metric.

$$T = T_e - T_s \tag{4.2}$$

  where T is the total execution time, T_e is the end time, and T_s is the start time of the analyzed part.

- **Speedup:** It presents the improvement obtained by our implementation when compared to the traditional SUMO simulation. Equation 4.3 describes this metric.

$$S = T_t/T_d \tag{4.3}$$

  where T_t is the execution time of the traditional SUMO simulation, T_d is the required time of our distributed implementation, and S is the ratio of traditional and distributed time.

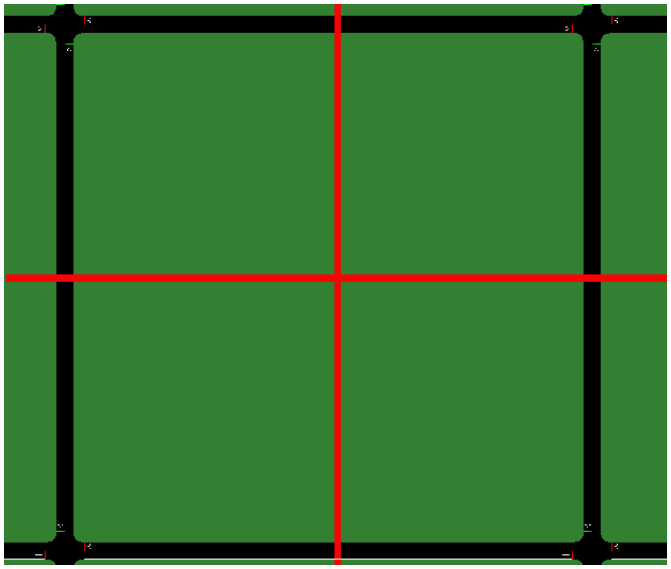


Figure 4.13: Network Partition for four nodes

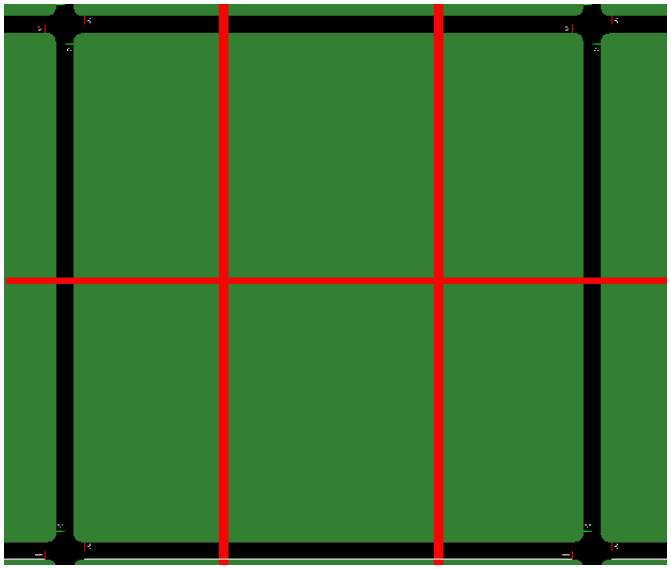


Figure 4.14: Network Partition for six nodes

Chapter 5

Results and Discussion

This chapter presents the results and discussion obtained from our implementation of a distributed SUMO. These results include several simulations performed in one, two, four, and six nodes of a distributed system with different vehicular densities. The discussion presents insight on the execution time while comparing different aspects of the solution.

5.1 Results

The previous chapter showed that our methodology could simulate traffic flow scenarios for several thousand vehicles. This simulation runs on top of the SUMO framework and takes advantage of TraCI and MPI tools to distribute the work. The results in this section provide insight into the behavior of the different simulation elements, particularly concerning time. We present an analysis of the time consumption of each one of the elements: simulation, TraCI, and MPI. The results include the output of simulations such as the one presented in Figure 4.12.

The results presented in Figure 5.1 include a GUI perspective of a vehicle in a partition near the border and transferred into another partition. Before the vehicle goes across the border, the leaving node sends the information to a receiving node with all the vehicle information. Once the communication completes, the receiving node will add the vehicle to this partition and start coexisting in both partitions. Figure 5.2 represents this behavior. Then, the back-controlling procedure explained in Chapter 4 is performed.

The border management performs the process of back-controlling vehicles by sending information back to the original partition. After several steps, when the vehicle leaves the actual partition, TraCI deletes it from the original partition to only exist in the target node. Figure 5.3 and 5.4 illustrate this process in a SUMO graphical interface.

After executing simulations, as detailed in the experimental setup, the results compare the time requirements of our system. The first results showed that our system presents increased execution time as the number of nodes grows. It was necessary to perform a simulation using TraCI actions in the single node execution to appreciate our implementation better. Figure 5.5 depicts this behavior. However, further analysis indicates that the time required to perform a simulation step decreases with more nodes. We compare it to the required SUMO simulation time to reinforce the previous results. This simulation

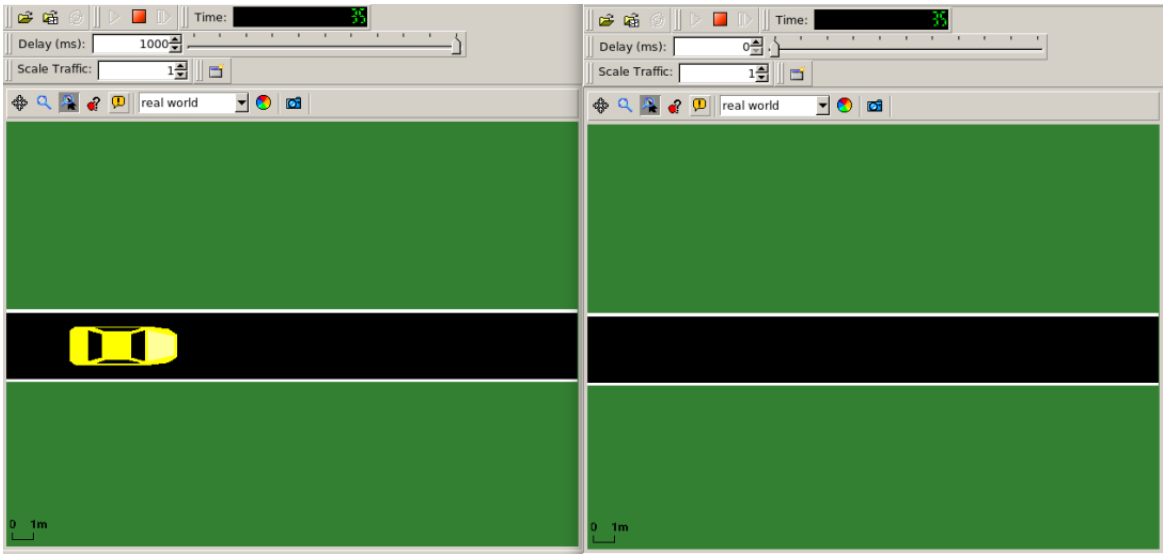


Figure 5.1: Vehicle in Partition A before arriving to the Partition B.

time presents an overview of SUMO calculation requirements, and Figure 5.6 illustrates this behavior.

The previously obtained results indicate that the system works properly by reducing the SUMO required simulation time. However, the overall result shows that an imposed overhead acts on top of SUMO, increasing the execution time performed by the system. Further analysis in this area required the implication of communication and TraCI aspects in the implementation. For this purpose, we perform a time analysis in terms of MPI synchronization and TraCI operations. Figure 5.8 depicts a relation between the number of vehicles in a simulation and the MPI synchronization time for the communication between nodes. Furthermore, Figure 5.9 presents a similar analysis as the previous one but in terms of TraCI time required to perform real-time operations within a partition. Finally, Figure 5.7 presents the maximum obtained speedup in SUMO execution time of our implementation.

The obtained results indicate an extensive overhead imposed by the communication and control processes during the distributed simulation. However, the SUMO calculations decrease as the number of nodes increase, achieving one of the project goals. In the following section, we will discuss our results.

5.2 Discussion

The results presented in the previous section provide insight into time requirements in the implemented SUMO distributed simulation. The three main elements of the system require different operation times that influence the overall performance of the implementation. We present a discussion regarding the obtained results with a focus on future research.

The total time of the simulation is depicted in Figure 5.5. We can see that the single-node simulation requires less time than the multiple nodes approach. Several factors can lead to this behavior. One crucial factor is the overhead imposed by the use of TraCI

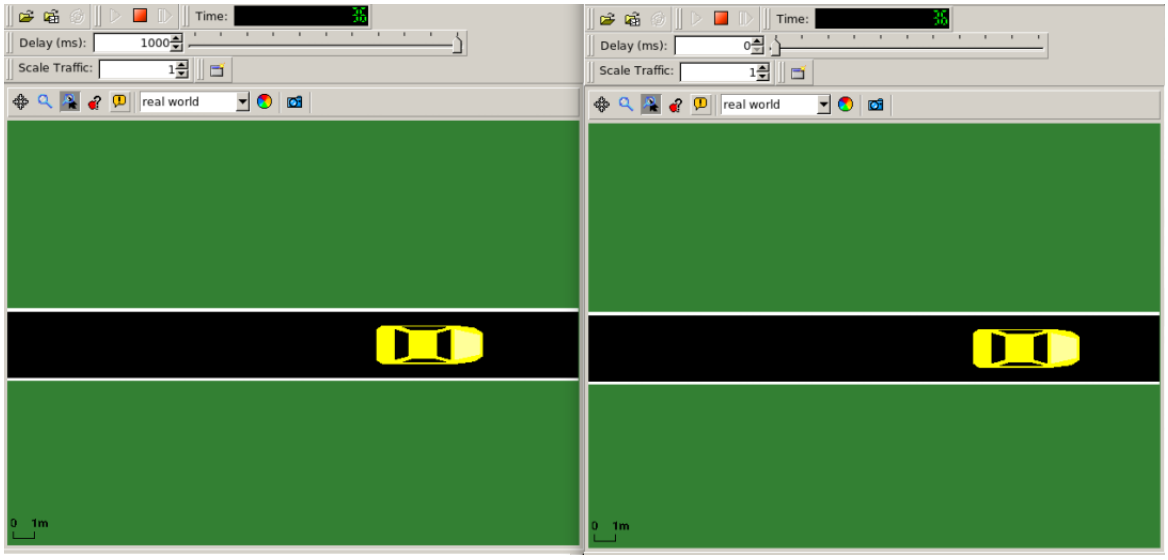


Figure 5.2: Vehicle in Partition A and B after transference.

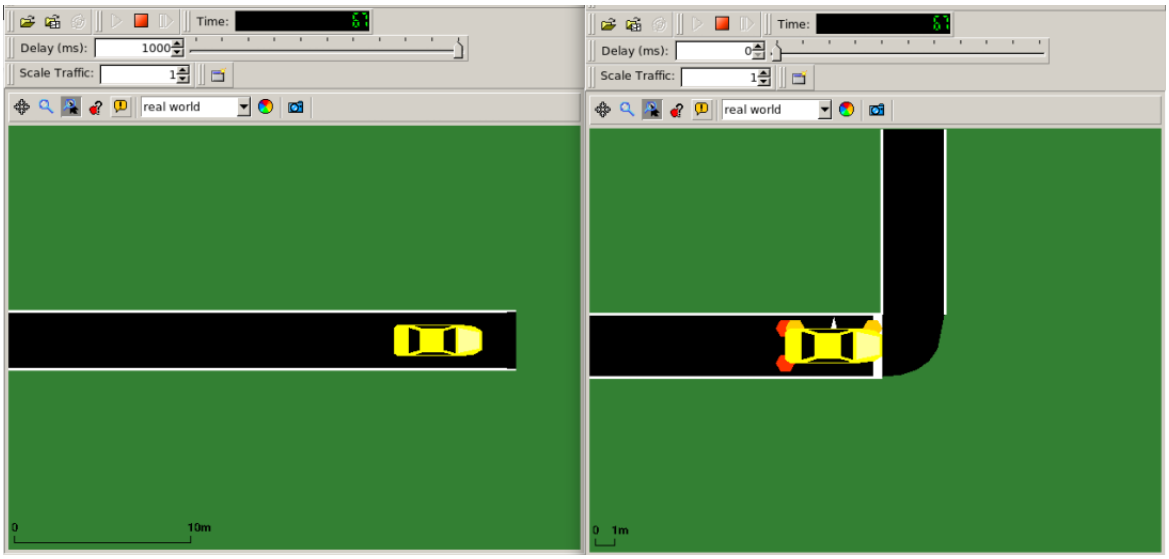


Figure 5.3: Vehicles coexist in both partitions after crossing the edge.

to change vehicular behavior in real-time. This leads to several computations needed to retrieve vehicle information and significantly add and remove vehicles within a partition. The other factor to consider is the overhead demanded by the synchronization time required to send information between nodes. The barrier used in MPI requires some partitions to inquire in lazy waiting until completing the communication. This problem can be solved using a more balanced simulation by controlling the number of vehicles present in each partition per time step and the number of vehicles transferred between adjacent nodes.

A crucial aspect of these results relies on the fact that the required SUMO simulation time reduces with more nodes in the distributed system. This implies that the required computations to perform a simulation step effectively reduces, and thus, the simulation time diminishes. Figure 5.6 presents an overview of the time required to perform a simula-

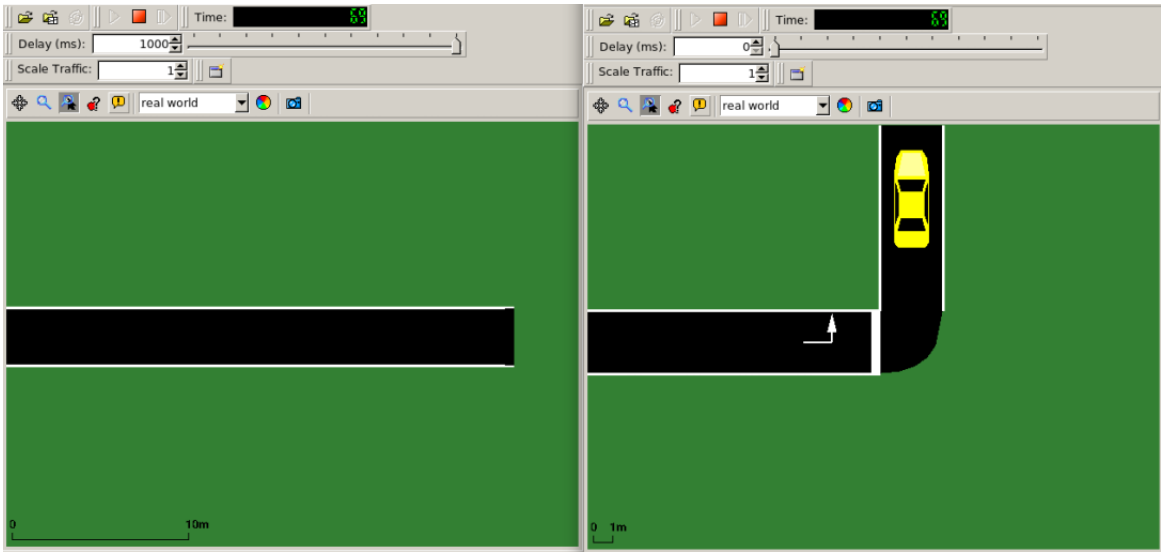


Figure 5.4: Vehicle in Partition A stop existing after leaving the border edge.

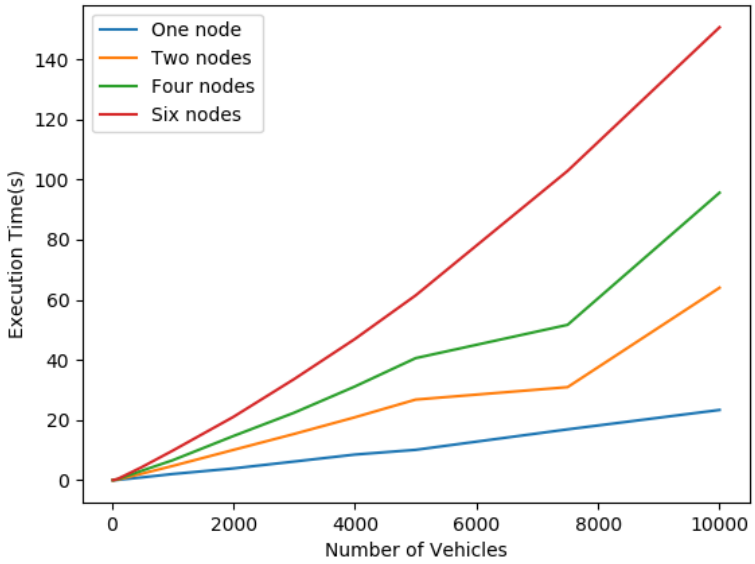


Figure 5.5: Total simulation time.

tion in the distributed system. We can observe that the required time reduces to less than 40% in six node scenarios compared to a standard simulation. These results also imply that our implementation effectively distributes the work by giving each SUMO partition less work to perform during a simulation. The tendency of this time seems to maintain during all the vehicular densities, and the augmentation of nodes reduces the simulation time, which evidences the reduced calculations performed by SUMO in our implementation. Further analysis of these results presents the obtained speedup in Figure 5.7, which shows that our implementation obtains up 2.7x speedup compared to the traditional SUMO simulation. This speedup grows as the number of nodes increase, demonstrating the potential

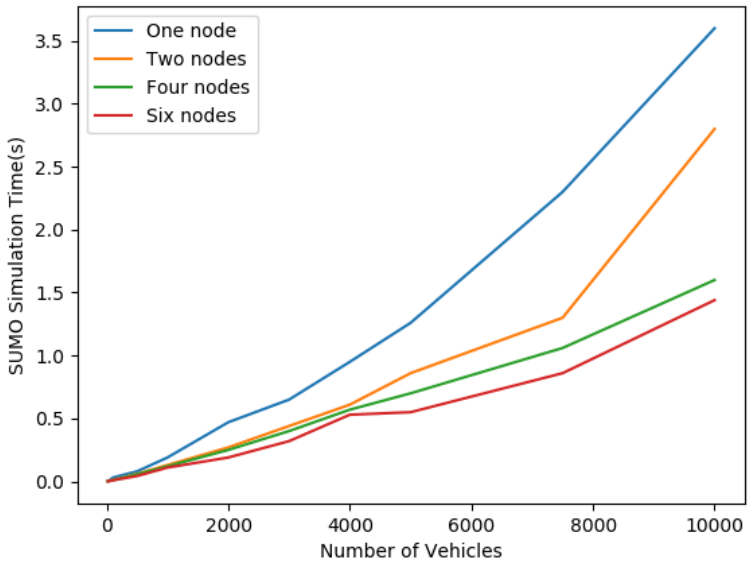


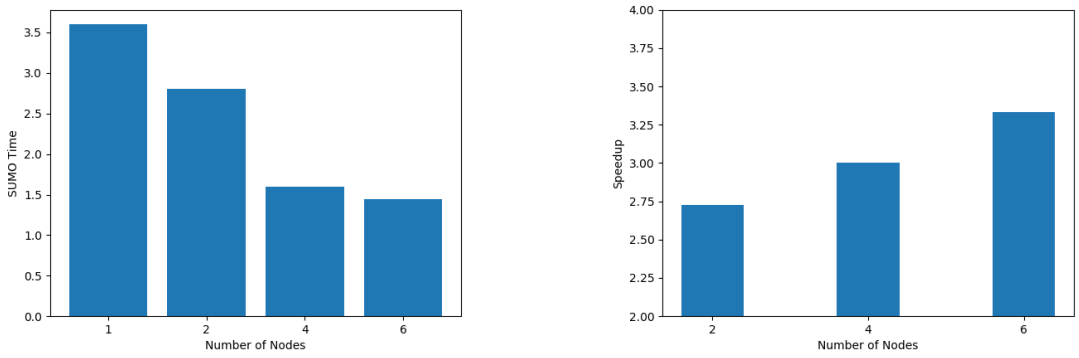
Figure 5.6: SUMO simulation time

scalability of our implementation.

As stated before, it is essential to analyze the overhead that MPI imposes in the communication environment of our implementation. This overhead results during the transfer of vehicles, or other information, between nodes. Figure 5.8 compares the required time to perform data transfers between nodes during the distributed simulation. We can see that this time rises as the number of nodes increases, indicating that the communication part of the implementation adds an undesired time to the output of the simulation.

The behavior in this communication between nodes causes more vehicle transfer as the number of partitions increase. The message transfer of a vehicle is demanding in communication time. As we have constant communication between nodes, due to the peer-to-peer topology of the implementation, it results in a lazy waiting behavior of the system while vehicles are being transferred. The inserted barrier contributes to synchronize all the nodes in the system. However, some nodes wait without performing any action, resulting in the imposed overhead. A solution for this problem includes performing communication in a defined number of steps, but this can jeopardize the simulation accuracy of SUMO. Another solution includes using a master-slave scheme that will reduce the number of messages sent between all nodes. However, this can add complexity to the implementation and the need for an extra node to act as central entity of the distributed system.

The most significant overhead is imposed by the control interface that performs real-time manipulation of the simulation via TraCI. Figure 5.9 depicts this time performance during the system execution. We can see that the time occupied by TraCI increases with the number of nodes. The interface will have to compute fewer vehicles, but the performed operations increase as more vehicles will be back-controlled and added/removed during the simulation. Another overhead includes the needed calculations to retrieve and compare the position of vehicles in entities that will possess more adjacent partitions. Figure 4.12 depicts the partition performed in two nodes. In this approach, each partition will calculate the



(a) Maximum Time (b) Speedup

Figure 5.7: SUMO speedup

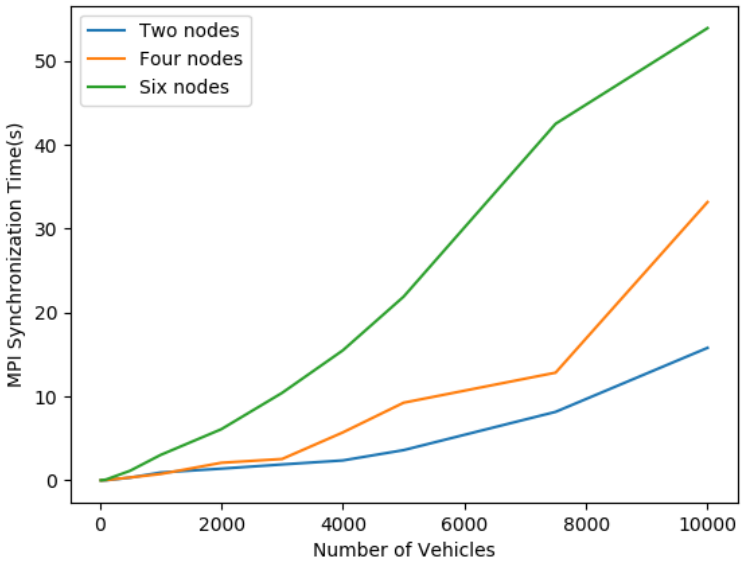


Figure 5.8: MPI synchronization time

border management for only one adjacent partition. In Figure 4.13 we have to manage the vehicular behavior for two adjacent partitions. Finally, in Figure 4.14 we have to manage the vehicle behavior for three adjacent partitions in some cases. For our partition algorithm, this number results in four adjacent partitions in more extensive scenarios.

The interface of TraCI works as a client/server architecture with TCP/IP communication. This can increment significantly the time required to manipulate simulation objects during the execution. A solution for this problem might include a control interface implementation directly in SUMO source code that will perform the same operations ignoring the TraCI communication overhead. This could improve the system performance but increase the level of complexity in the implementation.

Besides, more solutions can result in positive outcomes for future research. The use of

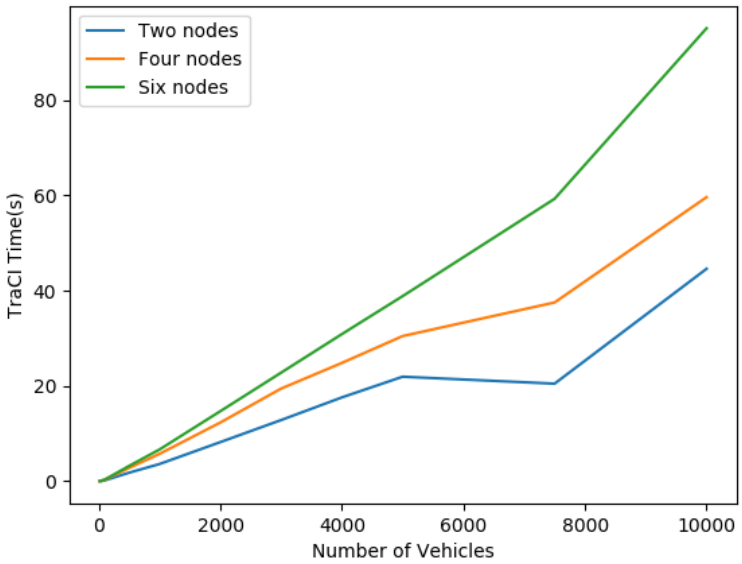


Figure 5.9: TraCI operations time

more balanced route data can result in a better distribution of work. However, it will not guarantee success in real scenarios as they will present unbalanced data. Another partition algorithm may lead to an efficient partition for both network and route files, causing a harmonization of the simulation elements. The implementation of an alternative to MPI may result in reduced synchronization time requirements. An interesting approach could implement OpenMP techniques in the source code to perform parallelism in SUMO calculations. Finally, using a lower-level language, such as C++, can benefit the performance of the system.

Chapter 6

Conclusions

This chapter presents our conclusions about implementation, results, analysis, recommendations, and future research ideas related to distributed traffic flow simulation.

6.1 Conclusions

The implementation of MPI techniques with TraCI facilitated the execution of distributed simulations using the SUMO package. A uniform space partition algorithm was used to divide the network and route files. In our proposal, each partition executes in a different node in parallel. Border management is essential to guarantee the correct behavior of the simulation. Asynchronous communication ensures proper communication between nodes. We used message passing to send vehicles between the different partitions. The border management found in the literature and described in Chapter 4 guarantees the accuracy of the implementation as it showed an accurate match between simulations running in different nodes.

The distributed simulation showed an increased total simulation time over the centralized simulation. However, each partition calculation was reduced, which showed that the time required to perform a simulation step decreases with more nodes in the system. This implies that the simulation time is effectively reduced in this implementation while maintaining its microscopic behavior.

The utilization of the MPI library and TraCI generated a significant overhead to the simulation. This overhead affected the overall performance of the simulation. In the analysis, the overhead increments due to the number of actions that TraCI must perform to retrieve the position of the vehicle at each time step and the real-time manipulation during the simulation. Another essential aspect to consider in the overhead is the MPI synchronization time required to send and receive information between nodes.

An essential aspect of the results showed the overhead imposed by the MPI implementation of the simulation. It evidenced a significant time added to the execution. This analysis showed that the time required to send a vehicle between nodes is significant, which causes a lazy-waiting in nodes that are not sending vehicles due to the barrier established in the synchronization process. It also exhibited an increased time influence regarding the network topology, which can cause the communication between more nodes in more

extensive scenarios.

Furthermore, TraCI presented the most considerable overhead in the implementation. Our experiments showed the impact of TraCI on the execution time of the distributed system. The analysis resulted in more border management calculations in a six-node scenario compared to smaller distributions.

Finally, the obtained results evidenced the fulfillment of the proposed objectives. We implemented an algorithm to partition the networks and routes. Then, we used these partitions to simulate them in a distributed environment using SUMO, MPI, and TraCI. Different SUMO simulations allowed the performance analysis of the implementation.

The results presented the time requirements of the proposed implementation and compared them to traditional simulation results. Although the execution time exceeded the expected results, our implementation showed that it could reduce the simulation calculations in each node. Further research can help to keep improving the performance of our implementation.

6.2 Recommendations

In this section, we introduce several recommendations regarding the problems and limitations of this implementation. These recommendations are the result of our learning process during the implementation of this work.

- Use more balanced data in the partitions. For example, each partition will have the same number of vehicles, and they will be equally transferred in the same time step.
- Perform a vehicle transfer in a different time step. For example, a node will communicate with others every two-time step. However, this approach can result in an accurate effect.
- Use the previous approach to back-control a car. This can cause less impact on the simulation performance and potentially decrease execution time.
- Implement a master-slave scheme that will reduce the message passing and increase the node requirements.
- During the algorithm design phase, we recommend using functions to implement the system. This technique enables a better structure and code reusability in other implementations.
- For the implementation of the system, we recommend using more nodes in the distributed simulation. This will allow us to have a better analysis of our approach.
- In the partition phase, we suggest using some graph partition algorithms. This could present a better insight into the results of our implementation.

6.3 Future Work

In this section, we propose approaches of implementation, which could result in future works.

- More balanced simulations can be implemented to avoid lazy waiting during the synchronization process, and this will increase the performance of the simulation.
- A graph space algorithm will be implemented to ensure a better distribution between partitions in more significant network scenarios and increase the workload distribution.
- The usage of another language, such as C++, could decrease the required time to perform the distributed simulation. We believe this could potentially increase the performance of the system.
- Work with an alternative to TraCI, developed in the SUMO source code that will control the vehicular behavior of the simulation but directly during the simulation step. This will lead us to omit the necessity of an intermediary such as TraCI that can add an undesired overhead to the simulation.
- An MPI alternative, such as OpenMP, implemented directly in SUMO source code to calculate the microscopic characteristics of vehicles, can significantly decrease the overhead obtained by this implementation.

Bibliography

[1] F. van Wageningen-Kessels, H. Van Lint, K. Vuik, and S. Hoogendoorn, "Genealogy of traffic flow models," *EURO Journal on Transportation and Logistics*, vol. 4, no. 4, pp. 445–473, 2015.

[2] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J. P. Hubaux, "TraCI: An interface for coupling road traffic and network simulators," *Proceedings of the 11th Communications and Networking Simulation Symposium, CNS'08*, pp. 155–163, 2008.

[3] A. Skjellum, N. E. Doss, and K. Viswanathan, "Inter-communicator extensions to mpi in the mpix (mpi extension) library," *Submitted to ICAE Journal special issue on Distributed Computing*, 1994.

[4] M. S. Ahmed and M. A. Hoque, "Partitioning of urban transportation networks utilizing real-world traffic parameters for distributed simulation in SUMO," *IEEE Vehicular Networking Conference, VNC*, pp. 1–4, 2017.

[5] A. Ventresque, Q. Bragard, E. S. Liu, D. Nowak, L. Murphy, G. Theodoropoulos, and Q. Liu, "Spartsim: A space partitioning guided by road network for distributed traffic simulations," in *2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*. IEEE, 2012, pp. 202–209.

[6] A. Acosta, J. Espinosa, and J. Espinosa, "Distributed Simulation in SUMO Revisited: Strategies for Network Partitioning and Border Edges Management," *Proceedings of the 4th SUMO User Conference*, no. July, pp. 61–71, 2016.

[7] Q. Bragard, A. Ventresque, and L. Murphy, "dSUMO: Towards a Distributed SUMO," *The first SUMO User Conference (SUMO2013)*, no. i, p. pp 132, 2013. [Online]. Available: http://hdl.handle.net/10344/3347

[8] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "SUMO–simulation of urban mobility: an overview," *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*, 2011.

[9] D. Smith, S. Djahel, and J. Murphy, "A SUMO based evaluation of road incidents' impact on traffic congestion level in smart cities," *Proceedings - Conference on Local Computer Networks, LCN*, vol. 2014-November, no. November, pp. 702–710, 2014.

[10] F. Malik, H. A. Khattak, and M. Ali Shah, "Evaluation of the impact of traffic congestion based on SUMO," *ICAC 2019 - 2019 25th IEEE International Conference on Automation and Computing*, no. September, pp. 1–5, 2019.

[11] J. Dargay, D. Gately, and M. Sommer, "Vehicle ownership and income growth, worldwide: 1960-2030," *The energy journal*, vol. 28, no. 4, 2007.

[12] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[13] B. D. Greenshields, "A study in highway capacity," *Highway Research Board Proc., 1935*, pp. 448–477, 1935.

[14] S. Krauss, "Microscopic modeling of traffic flow: investigation of collision free vehicle dynamics," *Forschungsbericht - Deutsche Forschungsanstalt fuer Luft - und Raumfahrt e.V.*, no. 98-8, 1998.

[15] L. A. Pipes, "An operational analysis of traffic dynamics," *Journal of applied physics*, vol. 24, no. 3, pp. 274–281, 1953.

[16] M. Bando, K. Hasebe, A. Nakayama, A. Shibata, and Y. Sugiyama, "Structure stability of congestion in traffic dynamics," *Japan Journal of Industrial and Applied Mathematics*, vol. 11, no. 2, p. 203, 1994.

[17] M. Bando, K. Hasebe, A. Nakayama, A. Shibata, and Y. Sugiyama, "Dynamical model of traffic congestion and numerical simulation," *Physical review E*, vol. 51, no. 2, p. 1035, 1995.

[18] R. Barlovic, L. Santen, A. Schadschneider, and M. Schreckenberg, "Metastable states in cellular automata for traffic flow," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 5, no. 3, pp. 793–800, 1998.

[19] S. C. Benjamin, N. F. Johnson, and P. Hui, "Cellular automata models of traffic flow along a highway containing a junction," *Journal of Physics A: Mathematical and General*, vol. 29, no. 12, p. 3119, 1996.

[20] A. Pentland and A. Liu, "Modeling and prediction of human behavior," *Neural computation*, vol. 11, no. 1, pp. 229–242, 1999.

[21] M. Brackstone and M. McDonald, "Car-following: a historical review," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 2, no. 4, pp. 181–196, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S136984780000005X

[22] P. G. Gipps, "A model for the structure of lane-changing decisions," *Transportation Research Part B: Methodological*, vol. 20, no. 5, pp. 403–414, 1986.

[23] U. Sparmann, "Spurwechselvorgänge auf zweispurigen bab-richtungsfahrbahnen," *FORSCH STRASSENBAU U STRASSENVERKEHRSTECH 263*, no. 263, 1978.

[24] W. Leutzbach and F. Busch, "Spurwechselvorgänge auf dreispurigen bab-richtungsfahrbahnen," *Institut für Verkehrswesen, Universität Karlsruhe*, 1984.

[25] D. Chowdhury, D. E. Wolf, and M. Schreckenberg, "Particle hopping models for two-lane traffic with two kinds of vehicles: Effects of lane-changing rules," *Physica A: Statistical Mechanics and its Applications*, vol. 235, no. 3-4, pp. 417–439, 1997.

[26] A. Latour, "Simulation von zellularautomaten-modellen für mehrspurverkehr," *Schriftliche Hausarbeit im Rahmen der Ersten Staatsprüfung*, 1993.

[27] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wiessner, "Microscopic traffic simulation using sumo," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2575–2582.

[28] D. Krajzewicz, G. Herktorn, C. Rössel, and P. Wagner, "SUMO (Simulation of Urban MObility)-an open-source traffic simulation," *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*, pp. 183–187, 2002.

[29] D. Krajzewicz, M. Bonert, and P. Wagner, "The open source traffic simulation package SUMO," *RoboCup 2006 Infrastructure Simulation Competition*, pp. 1–5, 2006. [Online]. Available: http://en.scientificcommons.org/20058515

[30] K. W Axhausen, A. Horni, and K. Nagel, *The multi-agent transport simulation MATSim.* Ubiquity Press, 2016.

[31] J. H. Banks, *Introduction to transportation engineering.* McGraw-Hill, 2002.

[32] S. Janz, "mikroskopische minimalmodelle des straßenverkehrs "," Ph.D. dissertation, Diploma Thesis, 1998.

[33] J. Erdmann, "Lane-Changing Model in SUMO," *SUMO2014 Modeling Mobility with Open Data*, pp. 77–88, 2014.

[34] M. P. Forum, "Mpi: A message-passing interface standard," 1994.

[35] J. Dongarra *et al.*, "Document for a standard message-passing interface," in *Message Passing Interface Forum*, 1993.

[36] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Computing*, vol. 22, no. 6, pp. 789–828, 1996.

[37] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting.* Springer, 2004, pp. 97–104.

[38] W. Gropp and E. Lusk, "User's guide for mpich, a portable implementation of mpi," 1996.

[39] L. Clarke, I. Glendinning, and R. Hempel, "The MPI Message Passing Interface Standard," *Programming Environments for Massively Parallel Distributed Systems*, vol. 6643, pp. 213–218, 1994.

[40] L. Dalcín, R. Paz, and M. Storti, "MPI for Python," *Journal of Parallel and Distributed Computing*, vol. 65, no. 9, pp. 1108–1115, 2005.

[41] G. Karypis and V. Kumar, "Analysis of multilevel graph partitioning," in *Supercomputing'95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing*. IEEE, 1995, pp. 29–29.

[42] G. Karypis and V. Kumar, "Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," 1997.

[43] C. Mayer, M. A. Tariq, R. Mayer, and K. Rothermel, "GrapH: Traffic-Aware Graph Processing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1289–1302, 2018.

[44] A. Steed and R. Abou-Haidar, "Partitioning crowded virtual environments," in *Proceedings of the ACM symposium on Virtual reality software and technology*, 2003, pp. 7–14.

[45] Z. Fu, "GeoSparkSim: A Scalable Microscopic Road Network Traffic Simulator Based on Apache Spark," Ph.D. dissertation, GeoSparkSim: A Scalable Microscopic Road Network Traffic Simulator Based on Apache Spark by Zishan Fu A Thesis Presented in Partial Fulfillment of the Requirements for the Degree Master of Science Approved April 2019 by the Graduate Supervisory Committee:, 2019.

[46] "A new strategy for synchronizing traffic flow on a distributed simulation using SUMO," *EPiC Series in Engineering*, vol. 2, pp. 152–141, 2018.

[47] T. Potuzak, "Distributed-parallel road traffic simulator for clusters of multi-core computers," *Proceedings - IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pp. 195–201, 2012.

[48] F. Tesser, "Distributed message passing with mpi4py," in *Euroscipy 2016*, 2016.