Universidade Federal
do Rio de Janeiro
Escola Politécnica

PROPOSTA E AVALIAÇÃO DE UM SISTEMA DE TRANSPORTE COM COORDENAÇÃO DISTRIBUÍDA

Amanda Camacho Novaes de Oliveira

Projeto de Graduação apresentado ao Curso de Engenharia de Controle e Automação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Amit Bhaya

Rio de Janeiro
Janeiro de 2020

PROPOSTA E AVALIAÇÃO DE UM SISTEMA DE TRANSPORTE COM COORDENAÇÃO DISTRIBUÍDA

Amanda Camacho Novaes de Oliveira

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO DE CONTROLE E AUTOMAÇÃO.

Examinado por:

A. Bhaya.

Prof. Amit Bhaya, Ph.D.

Prof. Marcos Vicente de Brito Moreira, D.Sc.

Prof. Daniel Ratton Figueiredo, Ph.D.

RIO DE JANEIRO, RJ – BRASIL
JANEIRO DE 2020

Camacho Novaes de Oliveira, Amanda

Proposta e avaliação de um sistema de transporte com coordenação distribuída/Amanda Camacho Novaes de Oliveira. – Rio de Janeiro: UFRJ/ Escola Politécnica, 2020.

XI, 86 p.: il.; 29, 7cm.

Orientador: Amit Bhaya

Projeto de Graduação – UFRJ/ Escola Politécnica/ Curso de Engenharia de Controle e Automação, 2020.

Referências Bibliográficas: p. 81 – 82.

1. Sistema de Transporte. 2. Transporte Automatizado. 3. Ride hailing. 4. Ride Sharing. 5. Transporte Distribuído. 6. Modelagem por Autômatos. 7. SAVs. I. Bhaya, Amit. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia de Controle e Automação. III. Título.

Agradecimentos

Começo agradecendo aos meus pais, Márcia e Izo. Pai, mãe, vocês são incríveis, sempre estiveram comigo, sempre se esforçaram para me dar o melhor de tudo e me ensinaram muito. Se cheguei aqui foi por causa de vocês. Agradeço também aos meus irmãos, Tiago e Iasmin, que estiveram ao meu lado por toda a minha vida, nos bons e maus momentos, e foram grande atuadores na minha formação como pessoa. E agradeço ao Lukas, pelas experiências maravilhosas que vivemos juntos. Amo muito todos vocês.

À minha família, meus avós, tios e primos. Em especial agradeço aos meus primos Rafael, Leonardo Joia e Nádia, por terem me ajudado com este trabalho, mesmo que às vezes só para escutar eu divagar sobre o assunto.

Agradeço à Universidade Federal do Rio de Janeiro, por todas as oportunidades que me foram fornecidas. Agradeço aos professores, agradeço à MinervaBots, equipe de competição da qual fiz parte. Agradeço ao Laboratório de Processamento de Sinais, no qual fiz iniciação científica, e ao Werner, o Seixas e a Dayane, que me orientaram. Conjuntamente, agradeço ao CERN, pela oportunidade de trabalhar em uma organização internacional e ter tido uma experiência de vida ímpar. Aprendi muito nesses anos, e não me furto a dizer que não sou mais a mesma pessoa que começou o curso seis anos atrás.

Aos meus amigos, em especial à T-18, que me proporcionou momentos incríveis nesses anos de faculdade. Ao Braian, que me ajudou muito com este trabalho. À Amanda, minha xará. Ao Daiha, o maior zueiro que eu já encontrei. Ao Patusco, melhor cozinheiro de ECA. Ao Eric, dono dos comentários mais pertinentes (como o famoso *Caiu aí*). Ao Vinicius, com o qual este trabalho começou a se desenvolver. À Elisa, que praticamente se tornou agregada da turma.

Em especial agradeço ao meu orientador, Amit Bhaya, por todas as ideias, toda a ajuda, apoio e ensinamentos durante o desenvolvimento deste trabalho.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/ UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro de Controle e Automação.

PROPOSTA E AVALIAÇÃO DE UM SISTEMA DE TRANSPORTE COM COORDENAÇÃO DISTRIBUÍDA

Amanda Camacho Novaes de Oliveira

Janeiro/2020

Orientador: Amit Bhaya

Curso: Engenharia de Controle e Automação

Transporte é algo extremamente necessário em centros urbanos e, portanto, este é um tópico de pesquisa importante no mundo todo. Muitas inovações surgiram recentemente na área e muitas já se tornaram indispensáveis para habitantes de grandes metrópoles. Este trabalho apresenta uma proposta de sistema de transporta dedicado que se baseia em automação distribuída, ou seja, na qual a decisão de designação de veículos para clientes é realizada de maneira descentralizada.

O sistema é composto por um conjunto de veículos não tripulados de plano de vias fixas, com estações de embarque em posições especificas. Clientes acessam os veículos nessas estações através do envio de requisições de serviço, que solicitam transporte de uma estação de origem para uma de destino. Os veículos executam algoritmos de decisão distribuídos para determinar a alocação dos clientes, algoritmos esses que foram modelados por autômatos. Uma simulação do sistema proposto também foi desenvolvida e utilizada como plataforma para experimentos e para otimizar a operação do sistema.

Abstract of Undergraduate Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Engineer.

PROPOSAL AND EVALUATION OF A TRANSPORTATION SYSTEM WITH DISTRIBUTED COORDINATION

Amanda Camacho Novaes de Oliveira

January/2020

Advisor: Amit Bhaya

Course: Automation and Control Engineering

Transportation in urban centers is greatly needed and thus an important research topic worldwide. Several innovations have arisen recently in this area and have already become indispensable to the inhabitants of large metropolitan cities. This work presents a proposal for a dedicated transportation system that relies on distributed automation, meaning that decision making in terms of assigning vehicles to clients is carried out in a decentralized manner.

The system is composed of a set of autonomous vehicles using routes with a fixed layout, with stations specified on each route. Clients access the vehicles at these stations by sending out requests to be taken from origin station to destination station. Vehicles execute distributed decision making algorithms, modeled by automata, to allocate themselves to clients. A simulation engine of the proposed system was also developed, and used as a platform for experiments as well as for optimization of the operation of the system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Public transportation systems are of great interest worldwide. There are many transportation models being implemented nowadays. In the past there were only taxis and public transport in the form of bus and train-like lines. Nowadays, however, many other options are being implemented and encouraged. Some examples include bike renting in the urban centers, for example, like Bike Itaú in Rio de Janeiro, but also systems of ride hailing, ride sharing, car sharing, etc., like Uber, Waze Carpool, Grab, Lyft, Car2Go, etc. [1]

A new revolution in the field of transportation is likely to occur with the implementation of shared autonomous vehicles (SAVs), which could bring competitive advantages in the form of reduced cost and possibly added convenience [2]. Systems with SAVs are now being extensively studied and implemented and have long ceased to be science fiction. With SAVs, the service cost does not need to include human labor like the aforementioned systems, and neither does the user need to make a trip to access and return the vehicle.

This work describes an distributed autonomous transportation system, with the use of smart vehicles and no centralized dispatch system. This system is dedicated, constructed in the image of taxi services, with client requests happening in real time and with random origin and destination locations. The system conceived is based on the theory of distributed algorithms, which is detailed in [3]. Specifically, the Bully Algorithm [4] for distributed leader election served as inspiration for the

variant implemented in this work, as will be further discussed.

The project implements a program that simulates the system operation and gives as output the vehicles and clients behaviors. This can serve as basis for the implementation of the real system, enabling optimization of the system parameters, and verifying possible failure cases. The metrics used to analyze said system are based on previous work on the field, like [5], which implements railways timetable determination optimization, as well as [6], which optimizes a vehicle dispatch system. The simulation code implemented can be seen in GitHub© repository [7].

The following document is organized in a system description in chapter 2, a description of the solutions given for each aspect of the proposed system in chapter 3, a description of the technical aspects of the simulation implementation in chapter 4, the simulation analysis and experiments results in 5 and lastly the concluding remarks in 6.

Chapter 2

Distributed automation of a transportation system

The work hereby described envisions an autonomous transportation system that is capable of servicing individual passengers. It does so in real time, and without the use of a dispatch center, with decentralized coordination, while meeting certain performance measures, which will be introduced later.

The system integrates a combination of characteristics of several different existing systems. It is assumed that there exists a fleet of vehicles. These can be thought of as trams, trains or some similar vehicle, because they run on a fixed grid, and only stop at certain specific locations in this grid, the stations. In this work, the term tram is chosen to represent this class of vehicles.

The system can also be thought of as a taxi service, implementing ride hailing, since the vehicles are designed to handle individual client requests. Passengers arrive at the stations and solicit transportation to other stations. These requests should be handled in a manner that is efficient, both from the passenger and the tram system point of view.

In order to build a system that is capable of handling of the requests without a centralized dispatch system, it is assumed that trams within a certain distance from each other are able to communicate amongst themselves. Furthermore, client

requests are broadcasted to all trams in its vicinity, reaching all vehicles within a circle of a certain radius, centered at the passenger.

2.1 Operation assumptions

The transportation system envisioned in this monograph operates under a series of assumptions. These assumptions will mold the designed system, and are elaborated in the following sections, subdivided into tram movement and request handling assumptions.

2.1.1 Tram Movement

Movement of trams is subject to the following rules:

1. The full map of tracks is available and known to all trams.
2. The tracks are assumed to be all bidirectional. There is no imposed direction for traffic.
3. Trams are capable of moving through the tracks in the grid autonomously, without the need for a human operator.
4. Trams are able to calculate the shortest routes between two given locations in the grid, without taking traffic into account.
5. Trams have the capacity to check for collision avoidance and can use extra platform areas – areas at stations or other predetermined places, where an unlimited number of trams can wait, without occupying any actual track – to wait for tracks to become available for use. It is assumed each track partition is able to solve the race conditions that will arise.

2.1.2 Client Request Management

When clients want to utilize the service they make a transportation request, broadcasting it within a certain radius, which meant that trams within this radius receive

the request.

All trams that receive a request start processing it, and attempt to arrive at a consensus as to which one of them is going to service the request. By the end of the processing, it must always happen that one tram – and one tram only – is elected as the winner of the competition between the trams to service the request.

2.2 Efficiency parameters

Requests should be handled efficiently, both from the passenger and from the system point of view. Therefore it becomes necessary to define efficiency parameters and thus to ascertain what is considered an efficient solution. For the mode of transportation implemented, the most important factors to be measured are the client waiting times and the overall system cost. [5][6]

The system cost depends on several variables as well as on the implementation of the system design. Clearly, the number of trams available will be a major influence on the overall costs, but there are several other variables that impact costs, which can only be estimated after all system details have been specified.

However, even without the cost equation, some measurements that will influence it, such as the distance traversed by the trams, which is proportional to the energy used by the system, are known to be important and therefore should be measured and recorded. Besides day to day operational costs, the tram mileage is also related to maintenance costs. Thus the total distance traversed is defined below as one of the key performance indicators, for the system operator.

Total Distance Traversed

The total distance traversed is the sum of the distances traversed by each tram during the some specified duration of the system operation.

For clients, quality of service is the most important aspect of system operation, followed by the cost of the service. Thus, one also needs to take into account some

measure of the service quality. The most obvious measure is the average waiting time, which refers to the average time a request takes to be dealt with. There are four times of interest in the system, described below. An illustration of these different time intervals and the relationship between them is shown in figure 2.1.

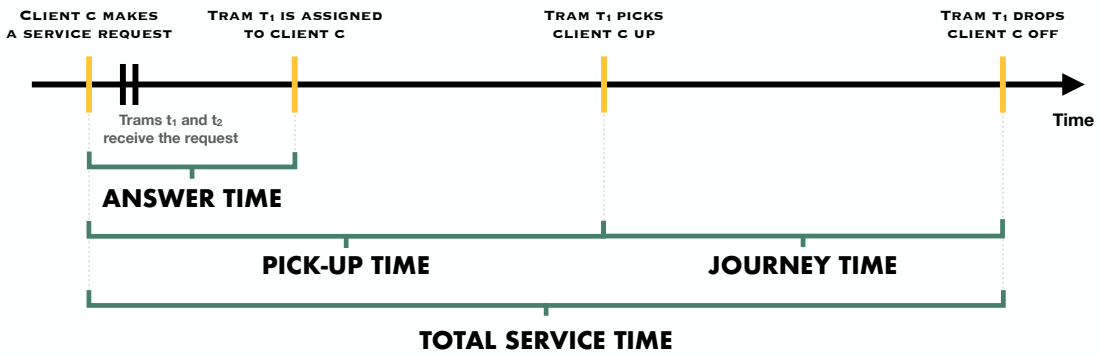


Figure 2.1: Visual representation of the system time parameters. The *answer time* is the interval between the request and the receiving of a tram assignment, the *pick-up time* is the time until the tram reaches the client position to pick it up, and the *total service time* is the total time interval until the client gets its request serviced.

Answer Time

This is the time between a client requesting the service and receiving a tram assignment. When a request is made, the trams will take a certain amount of time to process it and decide which tram should be the one to answer.

Pick-up Time

The pick-up time corresponds to the time interval between the request and the arrival of the assigned tram to the station where it is going to pick up the client. Clearly it must be greater than or equal to the answer time.

Journey Time

The journey time corresponds to the time interval between the client boarding the tram and being delivered at her destination.

Total Service Time

The total service time is the entire time elapsed between the client request and arrival at the destination. Thus it corresponds to the sum of the pick-up time and the journey time.

The overall design process is elaborated keeping in mind cost minimization and service quality maximization, using the definitions introduced above.

Chapter 3

Building the dedicated tram system model

This chapter describes the steps that lead up to the complete dedicated tram system model that uses distributed decision making. Section 3.1 describes the routing algorithm. Section 3.2 details the message exchange characteristics for client-tram and tram-tram communications, while section 3.3 shows how the so-called bully algorithm is modified for use as the distributed election algorithm. Finally, all these elements are combined into the overall system model in section 3.4.

3.1 Routing algorithm

One of the key assumptions of the system is that trams are autonomously capable of calculating their own routes while traversing the grid in response to client requests.

In order to allow trams to calculate their routes they must implement a routing algorithm. Dijkstra's Shortest Path First Algorithm[8] for finding the shortest path between nodes in a graph was chosen in this work. The algorithm iterates through the set of nodes in the graph, checking the minimum distances to get to them from the starting node until all nodes and associated edges have been verified. An illustration of the algorithm behavior can be seen in figure 3.1. The pseudo-code is detailed in appendix A.

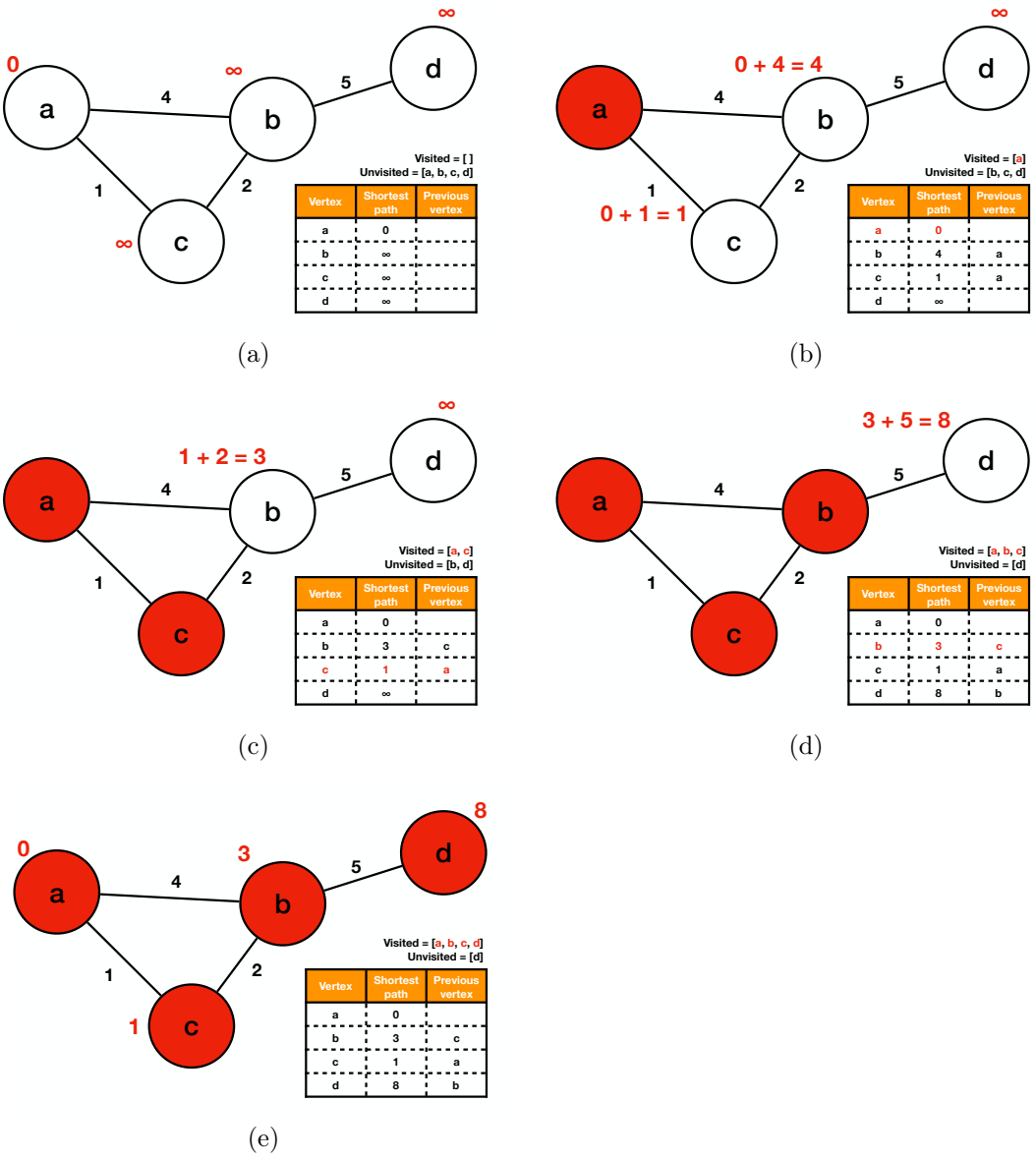
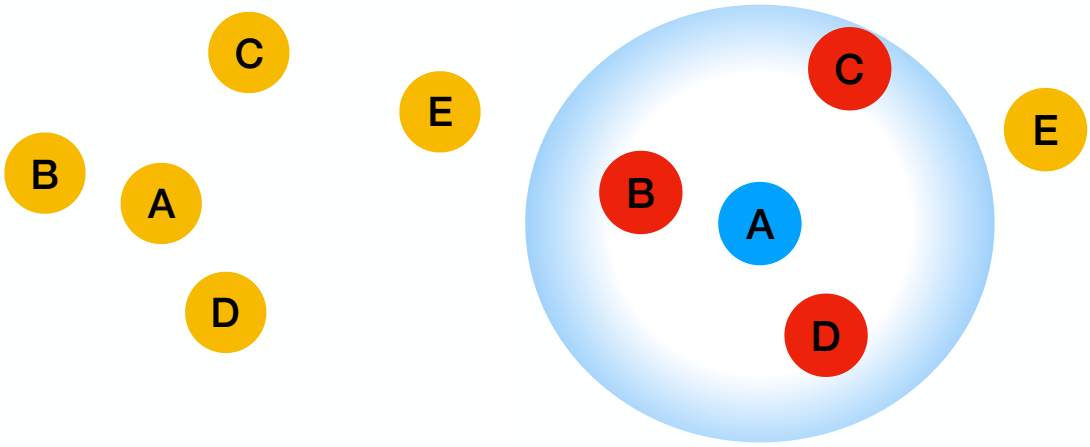


Figure 3.1: Example of the Dijkstra's Shortest Path First Algorithm for finding the shortest path from vertex a to d in the graph. Each figure shows one step of the algorithm, with the table next to the graphs specifying the known distances from vertex a. In figure (a) no vertexes have been checked and the only known distance is to the vertex a itself. Figure (b) shows the verification of vertex a's edges. Vertex c is the next to be checked in figure (c), for it has the smallest known distance from the unvisited vertexes. Figure (d) shows the analysis of vertex b. No further analysis is needed, because vertex d is reached and it is the desired destination. Figure (e) shows the algorithm end result.

3.2 Message exchange between trams or clients

As mentioned in chapter 2, the system conception assumes that trams and clients are able to communicate. Messaging is assumed to be in single-hop broadcast mode,

for both tram-tram and client-tram communication. Specifically, for any kind of message sent, a tram or client will broadcast it to its neighborhood, and those trams and clients within range will receive the message. This range is assumed to be a circle of specified radius centered on the sender. The processing of messages received is referred to as message handling and discussed in section 4.2.



(a) Set of agents in their initial states (b) System after agent A sends a message

Figure 3.2: Schematics of the proposed broadcast communication model. In figure (a) all nodes are in their initial states, which is represented by the color yellow. In figure (b), agent A has sent a message, which is represented by the blue color, and the blue circle around it corresponds to the message range. Agents B, C and D, that are within the broadcast range, receive the message – represented in red. However, agent E is outside reach, and thus does not receive the message transmitted, not changing to color red.

Broadcast range is clearly an important parameter: if it is too small, a client request may not reach any tram, and if it is too large, trams that are very far away will unnecessarily enter an election and increase the response time. The effects of this parameter will be discussed further and evaluated through simulation experiments reported in chapter 5.

3.3 Election algorithm

An election algorithm [9] is a distributed process designed to select one agent as a *leader* amongst a set of agents. Such algorithms are typically used in distributed

decision making processes. Since one leader is chosen redundancy is avoided and exactly one tram is assigned to service each request.

This work presents a variation of the Bully Algorithm [4], adapted to select the tram that has the shortest path to service the client, thus minimizing the total service time. The procedure always starts with a request from a client. At that moment, trams within range receive the broadcast message from the client. These trams calculate their respective routes and obtain the distance to the client. In order to implement the algorithm four types of messages are defined:

- *request*: This is the message sent by the client to request the service from the system. It initializes the election procedure.

- *election*: This is the message which the trams will send to each other in order to compare distances and determine which tram has the shortest path to the client.

- *election acknowledgment*: With this message tram A informs tram B that tram's B distance to the client is larger than it's own. On receiving this message, tram B knows that it has lost this particular election process and makes itself available for other elections resulting from client requests posterior to tram A's message.

- *leader*: This message is sent at the end of the election to other trams, to let all of them know that the election has finished and the sending tram has reported itself as the winner.

It is important to notice that, in this implementation, it is assumed that once the request is made, the client will not change its mind and cancel the request. This is an artificial stipulation made to simplify the system, although it could be modified to add the cancellation option with the addition of a message of this kind.

Given a client request, each tram within range calculates its distance to the client and broadcasts it through an election type message. Any tram receiving such

a message first verifies that it is a message pertaining to the election that it is currently processing and, if so, proceeds to compare the value received with its own distance.[1] There are three possible outcomes from this comparison:

1. The tram has a larger distance than the received one. In this case, the tram realizes it is not the one that will be answering the request, and takes itself out of the election process.

2. The tram has a smaller distance than the received one. Being the one closer to the target, it considers that it is more qualified to answer the request, and therefore broadcasts an *election* message of its own, if it has not done so yet. Furthermore, it sends an *acknowledgment* message to the tram that sent the received *election* message, to let it know that there are closer trams to deal with the request and that it should withdraw from the election procedure.

3. The tram has the same distance as the one received. In this case the tie is broken by the ad hoc method of checking the tram identifier number, which is unique to each tram. The tram with the highest ID will be the one selected, and it will proceed with one of the actions of items 1 or 2 above, depending on this result.[2]

After a tram first sends an *election* message, it waits for a specified amount of time for an *acknowledgment* message. If none is received, it times out and considers itself the election winner. In this instance, the winning tram broadcasts a *leader* message, informing other trams that it has won, and they should take themselves out of the election, if they have not done so already. Also, the tram sends a message to the client, informing him it has been assigned to pick him up.[3]

The waiting time interval should be chosen so that it gives enough time for other trams in the election to process the received distance and give an answer, if they win

[1] If the message does not pertain to its election it is discarded, for it is irrelevant to the tram in question.

[2] Although the identifier is not relevant for the analysis of which tram would be the best tram to answer the request, the uniqueness of identifiers guarantees that of the winner.

[3] This message has not been included in the list of messages because it is not part of the election processing per se, although it is necessary for the system conception.

the comparison. Thus, in theory the *leader* message should not be needed. However, in a more realistic case, there could be message delays, or delay in reading them, so this message serves as an extra security, to make sure that only one tram will win the elections.

The pseudo-code of this election algorithm is given in appendix B.

3.4 Transportation system modeling

A formal description that could be used to verify the correct operation of the algorithm is discussed in this section. The first attempt at modeling the election process used Petri Nets, described in section 3.4.1 below. Since the Petri Net model proved to be somewhat unwieldy, an automata model was developed. Its description is given in section 3.4.3.

3.4.1 Petri Nets and tram Petri Net model

Petri Nets [10] are untimed models capable of representing a wide variety of discrete event systems. They manipulate events according to certain rules, modeling a system not with relation to a time frame, but by so called *events*, which are any kind of occurrence that happens instantaneously and causes a system to go from one state to another.

A Petri Net is a graph, that captures a lot of structural information about the system. The net has transitions, usually associated with events, and places, that are associated with the system state, and contain the information on whether a certain transition can occur. The nodes of the graph, referred to as places are marked with tokens, which enable or inhibit the occurrence of transitions. A simple Petri Net graph example can be seen in figure 3.3.

The modeling of a single tram behavior through the use of Petri Nets resulted in the graph observed in figure 3.4. In the figure, *req* corresponds to a request message, *elec* to an election message and *elec_ack* for an election acknowledgment message.

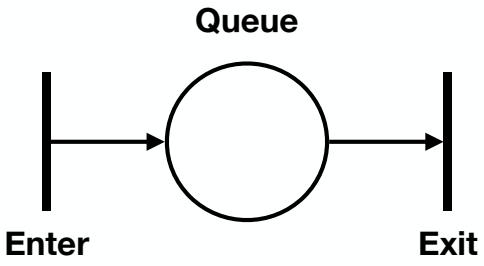


Figure 3.3: Simple Petri Net that represents the behavior of a queue. The transition to the left represents the entrance of a person in the queue. When it activates, one token is added to the queue, representing the person that is waiting. The transition on the left represents some person exiting the queue. It takes out one token from the queue and therefore can only be fired if there is at least one token in place, i.e. one person in the queue.

The leader message is denoted *leader*, without abbreviation. The model does not explicitly represent the sending of messages, but they are implicitly being sent in certain transition occurrences. For example, when *delay 2* is reached and the tram wins elections, there are messages that must be sent, according to the algorithm proposed.

Although a Petri Net represents well the behavior of a single tram, when attempting to extend this model to a representation of the global system behavior, the synchronization of multiple trams added a large number of places and transitions to the Petri net representation, and for this particular analysis it was rendered impractical.

The increased complexity of the Petri net representation of the simple scenario of two trams and a single client derived mainly from the coordination of actions between the trams due to exchanges, which necessitates a great amount of inhibitor arcs, and places to store said exchanges, since it was not possible to model both trams with a single group of places and represent each one by a token. Thus, it was decided to attempt the creation of a model based on automata, which turned out to be more intuitive to model the system considered.

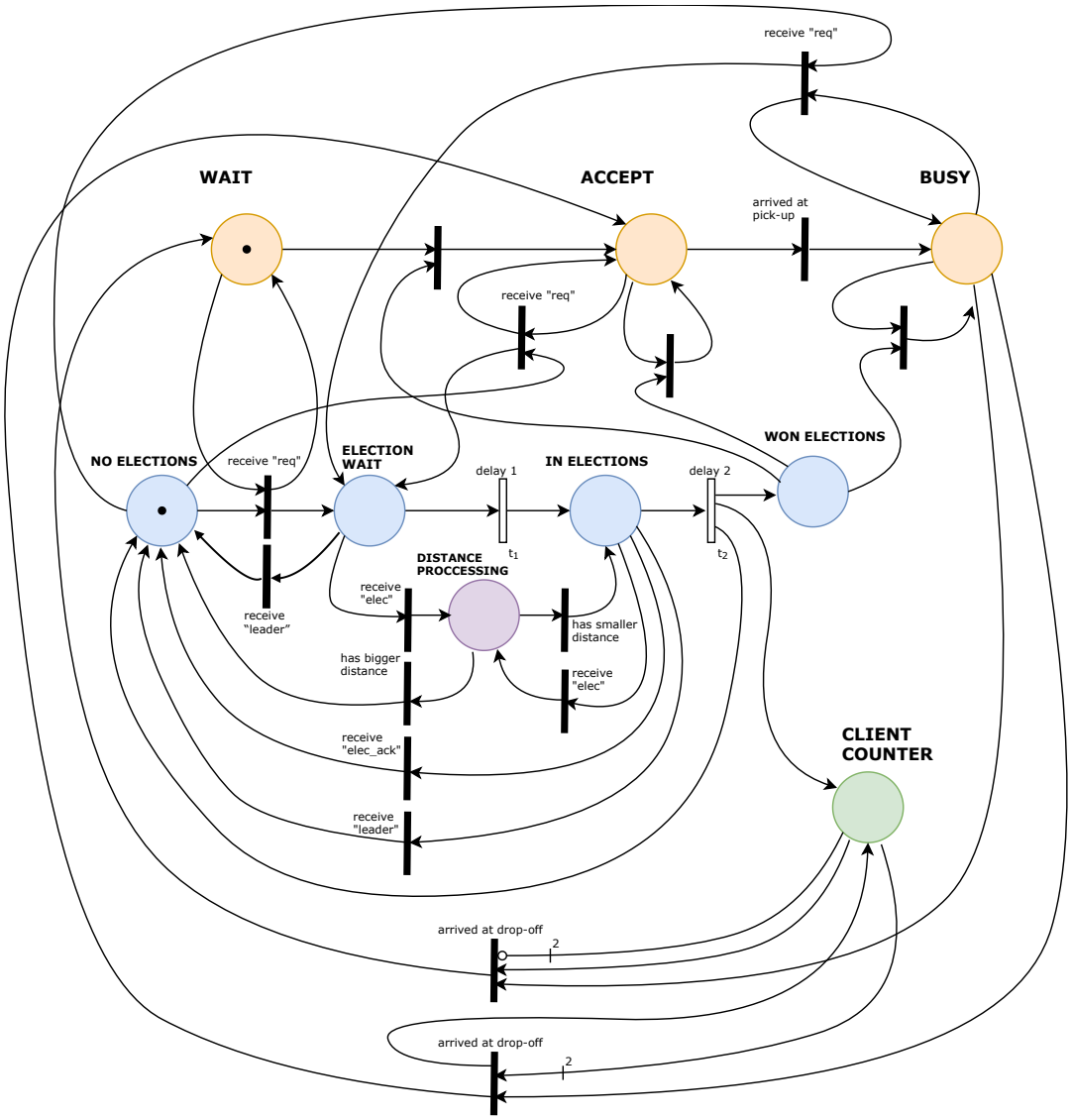


Figure 3.4: Petri Net of the tram behavior with respect to the election algorithm. The orange places correspond to the tram operational state. If it is in *wait*, it has no current clients to service. Being in *accept* means that the tram is going to its next client pick-up destination and has no clients in course. Lastly, *busy* signifies that the tram is currently with client. The blue places pertain to the election state. If in *no elections* the tram is not processing any request. If in *election wait* the tram has received a request, but not yet sent an election message. If in *in elections*, the tram has already broadcasted the election message and is awaiting the other trams answer. If in *won elections*, the tram has been selected to service the client in question. The auxiliary purple place represents the tram processing of a received election message. The remaining green place, the *client counter* keeps count of the number of clients that are assigned to be serviced by the tram.

3.4.2 Brief recap of automata theory

Automata [11] are discrete models of abstract computing devices or *machines*. They are a finite representations of a formal language, that might be an infinite set. Like

Petri Nets, they are capable of representing discrete event systems, but are a state machine: an automaton has a set of states, and for each state there are a set of events that can happen, making the automaton change the current state – sometimes going back to the same state as before.

A general deterministic automaton G can be defined as [12]:

$$G = (\chi, \Sigma, f, \Gamma, x_0, \chi_m) \tag{3.1}$$

- χ is the set of states.
- Σ is the alphabet, the set of events (each represented by a symbol) associated with the automaton G.
- $f : \chi \times \Sigma \to \chi$ is the transition function. $f(x, e) = y$ means that the occurrence of event e when G is in state x transitions the automaton to state y. This function is typically partially defined for its domain, which is to say, it is not defined for all pairs (x, e).
- $\Gamma : \chi \to 2^{\Sigma}$ is the active event function. The notation 2^{Σ} refers to the set of all subsets of Σ. $\Gamma(x)$, the active/feasible event set, consists in the set of all events e for which the function $f(x, e)$ is defined.
- x_0 is the initial state of the automaton, $x_0 \in \chi$.
- χ_m is the set of marked states. States are marked when they have a special meaning attached. Can be a "final" state, for example. It is a subset of the set of states: $\chi_m \subseteq \chi$.

The easiest way to represent an automaton is through its *state transition diagram*, which encapsulates the whole information contained in the six-tuple. This can be easily seen in the example of figure 3.5. This simple automaton G_b represents the behavior of an on/off swich, in which the state is changed each time the button is pushed.

- $\chi = \{\text{on}, \text{off}\}$
- $\Sigma = \{\text{push}\}$
- $f(\text{on}, \text{push}) = \text{off}$
- $f(\text{off}, \text{push}) = \text{on}$
- $\Gamma(x) = \text{push}\, \forall x \in \chi$
- $x_0 = \text{off}$
- $\chi_m = \emptyset$

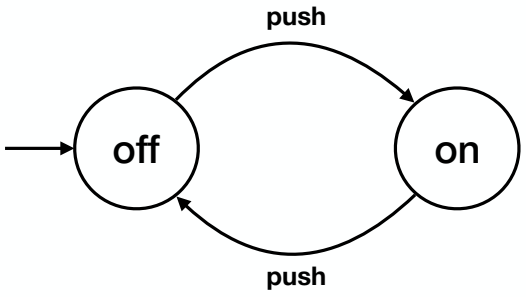


Figure 3.5: Automaton G_b, modeling an on/off switch. The states correspond to *on* and *off*, and the only event modeled is *push*, the action of having the switch pressed.

An automaton is capable of generating a language, which is comprised of the all the possible sequence of events that can occur for a given automaton. In the example of figure 3.5, the language generated is any sequence of *push* events. That is to say, the language generated is composed of no events – the ξ event –, one *push* event, or any number of these events:

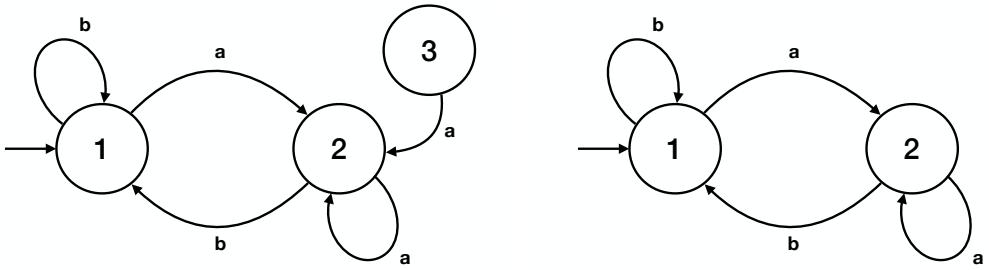
$$\mathcal{L}(G_b) = \{\xi, \text{push}, \text{push} - \text{push}, \text{push} - \text{push} - \text{push}, \cdots\}$$

With the definition of automata established, one can define operations that can be performed upon them. For the modeling of the transportation system the use of the *parallel composition* will be needed, and therefore it will be explained in detail ahead. However, this operation uses the *accessible part* of an automaton to be defined, and thus this operation will be discussed beforehand.

Accessible part of an automaton

The *Accessible Part* [12] is an operation over a single automaton, and corresponds to the removal of states that the automaton can never reach from the initial state x_0, that is to say, the states that are not *accessible* by some string in $\mathcal{L}$. When removing a state, all the transitions attached to it should also be deleted. This operation

performed upon an automaton G is represented by $Ac(G)$, and an example can be seen in figure 3.6.



(a) G, original automaton (b) $Ac(G)$, accessible part of automaton G

Figure 3.6: Illustration of the operation of obtaining the accessible part of an automaton. Figure (a), to the left, represents the original automaton G, and figure b, to the right, the accessible part of G. Note that neither state 3 nor its attached transitions are represented in $Ac(G)$, since this state could not be reached from initial state 1.

Parallel composition

Parallel composition [12] is used to combine two (or more) automata into a single one that represents the global behavior of both. Due to the way it handles private events between automata, the standard way of building models of entire systems out of models of individual system components is by parallel composition.

In general, when modeling systems that are formed by interacting components, each event set is formed by both private and common events. While the common events model the interaction between the components, the private ones pertain only to internal behavior. Therefore, while the common events must be synchronized and can only occur if both automata execute it at the same time, private events are not constrained and can be executed whenever possible.

Considering the two automata G_1 and G_2 so that:

$$\begin{cases} G_1 = (\chi_1, \Sigma_1, f_1, \Gamma_1, x_{01}, \chi_{m1}) \\ G_2 = (\chi_2, \Sigma_2, f_2, \Gamma_2, x_{02}, \chi_{m2}) \end{cases}$$

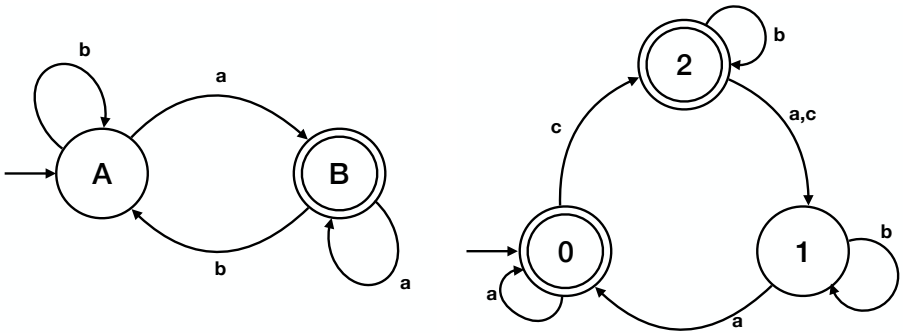
The parallel composition of G_1 and G_2, represented by $G_1||G_2$, is formally defined

as:

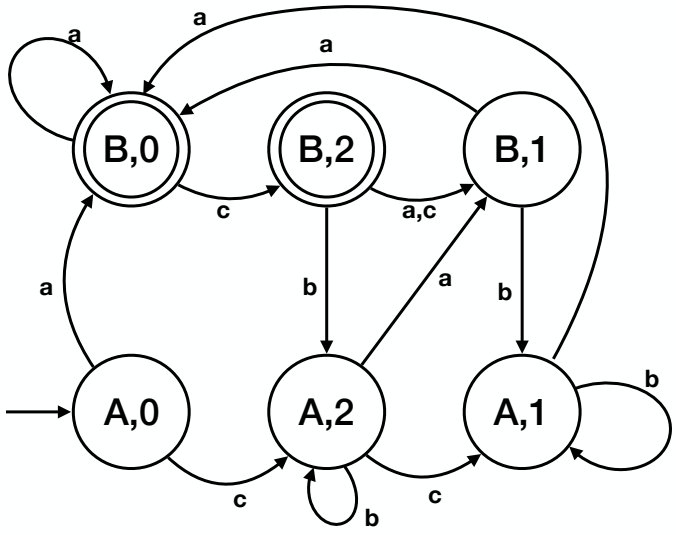
$$G_1||G_2 = Ac(\chi_1 \times \chi_2, \Sigma_1 \cup \Sigma_2, f, \Gamma_{1||2}, (x_{01}, x_{02}), \chi_{m1} \times \chi_{m2}) \tag{3.2}$$

$$\text{where} \quad f((x_1, x_2), e) := \begin{cases} (f_1(x_1, e), f_2(x_2, e)) & \text{if } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1, e), x_2) & \text{if } e \in \Gamma_1(x_1) \backslash \Sigma_2 \\ (x_1, f_2(x_2, e)) & \text{if } e \in \Gamma_2(x_2) \backslash \Sigma_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\text{and} \quad \Gamma_{1||2}(x_1, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \backslash \Sigma_2] \cup [\Gamma_2(x_2) \backslash \Sigma_1]$$



(a) Automaton G_1 (b) Automaton G_2



(c) Automaton $G_1||G_2$, the parallel composition of the other two

Figure 3.7: Illustration of the parallel composition operation between two automata. Figure (a) and (b), at the top, represent respectively automata G_1 and G_2, which are then combined through parallel composition in figure (c), obtaining the new atomaton $G_1||G_2$, which represents the combined behavior of the two previous ones together. Example extracted from [12].

The symbols $\cup$ and $\cap$ correspond, respectively, to union and intersection of sets. The operation $\chi_1 \times \chi_2$ is equivalent to all combinations (x_1, x_2) from sets χ_1 and χ_2, respectively. The symbol $\setminus$ represents an operation of removing elements that are part of a set from another set. So, for example, $\Gamma_1(x_1)\setminus\Sigma_2$ corresponds to the elements in $\Gamma_1(x_1)$ that are not in Σ_1, that is to say, $\Gamma_1(x_1) - \Gamma_1(x_1) \cap \Sigma_2$.

Figure 3.7 shows an illustrative example of the parallel composition operation.

3.4.3 Automata models for request handling

This section gives a formal model of the request handling and election procedure through the use of automata, previously seen in section 3.4.2.

The system has many variables that influence its behavior. For example, the more trams there are, the more messages exchanged for each request made. Clearly, the larger the client broadcast radius, the larger the number of trams that will receive the request, and of course the positions of the vehicles and clients themselves will also have to be taken into account. The frequency of client appearance – and consequently, the frequency of requests made – is also important, for a vehicle cannot start a second election procedure while in the middle of a previous one.

Therefore, in order to propose a model of the whole system, one needs to first specify what conditions will be taken into account. In this case, the modeling was started in the simplest possible scenario: with two trams, only one client at a time, and in which the broadcast radius covers the whole area that the trams traverse in. Also message receiving delays have been disregarded (as well as any errors in transmission, like message loss, receiving order error, etc.), considering a situation in which all participants receive any message sent instantly.

Automata are based on states and events, making them ideal for the model of this system: each message exchange can be seen as an event, and the trams and clients can act accordingly, changing their current state of operation, whenever such an exchange makes it necessary. Furthermore, with automata one can model the client and each tram separately, subsequently combining them into a global model of

the system using parallel composition. This feature makes modeling using automata simple and intuitive. Each step of this modeling will be thoroughly detailed below.

The models employ the use of marked states to analyze possible deadlocks, which would correspond to a failure in the system.

3.4.3.1 Model of the client interaction with the system

The client's role in the system is to make requests, and await an answer from the system. Each client submits its request on entering the system. Once it has been processed, one of the trams will have been selected to service it, and after the client is delivered to the requested destination, it leaves the system.

These states of operation are shown in the automaton of figure 3.8.

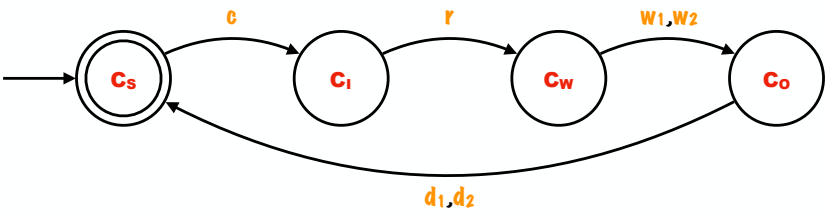


Figure 3.8: Automaton of the client interaction with the system, considering the situation of a single client at a time, and two trams processing its request. The initial state represents the situation of not having a client currently in the system. It changes state when a client arrives, and then again when the client makes a service request, initiating the request processing by the trams. By the end of this procedure, one of the trams will have been assigned and the automaton enters the operational state, in which the client is being serviced. Lastly, the system goes back to its initial state when the client is delivered.

As shown in the diagram, the *client status* is represented in 4 possible states:

- C_s: The initial state of the system, this means there are no current clients on the system. Therefore it also means that all previous clients have already been serviced, and therefore it is a marked state, for it is the state the system will always endeavor to return to.

- C_i: In this state, a client has entered the system, but not yet made any request.

- C_w: When in this state, the client has made its request and is waiting for the trams answer.

- C_o: This is the state where a client has been assigned a tram and is being serviced by it.

The alphabet of this automaton is composed of 6 symbols in total, each one corresponding to a specific event in the system. The definition is as follows:

$$\Sigma = \{\mathtt{c}, \mathtt{r}, \mathtt{w_1}, \mathtt{w_2}, \mathtt{d_1}, \mathtt{d_2}\} \tag{3.3}$$

- $\mathtt{c}$: This event corresponds to a client arrival.

- $\mathtt{r}$: Corresponds to a request being made. The request procedure involves the client itself sending a message to the trams in the vicinity, which receive said message and take appropriate action.

- $\mathtt{w}_i$: The $\mathtt{w}$ event signifies that the election procedure is over. If $\mathtt{w_1}$ happens the tram number 1 has won the election and will service this particular client, and if $\mathtt{w_2}$, tram number 2 has won.

- $\mathtt{d}_i$: The $\mathtt{d}$ event occurs when a client is delivered at its destination. In this case the client has been serviced and can therefore leave the system. $\mathtt{d_1}$ occurs when tram number 1 was the one servicing the client and $\mathtt{d_2}$ if it was number 2.

3.4.3.2 Tram model

The trams have to be able to process client requests and communicate among themselves in order to reach a consensus about which one is going answer the request. They do so through the election algorithm seen in section 3.3, which minimizes the total service time for the request. When the vehicles receive the request, they will calculate the route to delivery given their current positions and then compare the corresponding distances between themselves in order to arrive at a decision.

An example of the automaton model for the vehicle behavior is shown in figure 3.9. This diagram shows how vehicle number 1 acts in the system, and number 2 will have the same behavior, except that the events have the indexes *1* and *2* inverted.[4]

[4] This means that, for example, whenever vehicle 1 has a $\mathtt{e_1}$ event vehicle 2 will have $\mathtt{e_2}$, etc.

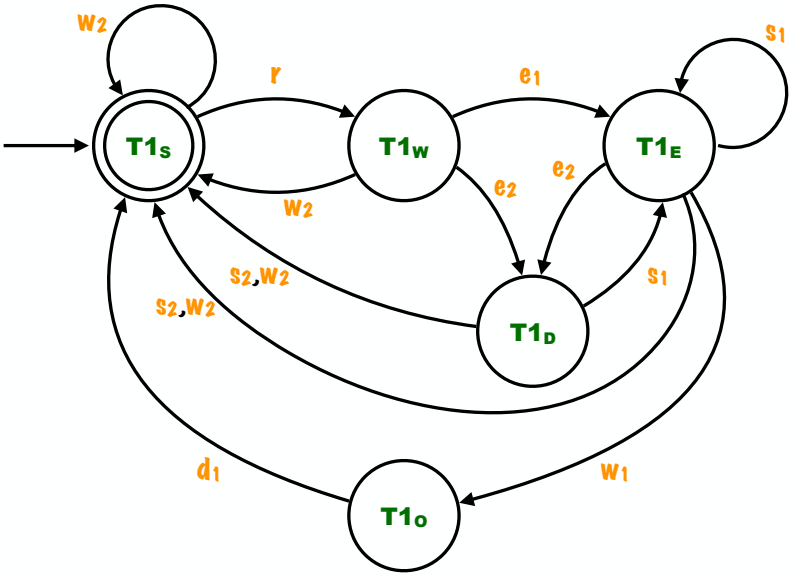


Figure 3.9: Tram automaton model (for tram number 1), considering the scenario of two vehicles and only one client at a time. In the initial state, there is no client request to be considered, and the tram is in wait mode. When it receives a request, it starts processing it. From there, it will either send or receive an election message, depending on who fires first. Whichever tram receives the message will compare the two distances (its own and the one received) and determine which one is the smallest. If it has the smallest distance, it will send an acknowledgment message to the tram that sent its distance in the first place (in this case the only other tram), to let it know it has lost the election, and send an election message to any other possible trams in the system.[5] If, however, the tram has the greater distance of the two it will take itself out of the election with no further action. Eventually the tram that was not taken out of the election will win the election and proceed to service the client. The cycle ends with the delivery of the client at the desired destination.

As the diagram shows, each tram has 5 states of operation. Each state is called Ti_x, where "i" corresponds to the tram number – in this case, number 1, the second tram would be 2 – and x is a letter that differentiates the different possible states the vehicle can be found in. These states are described as:

- Ti_s: The initial state in which the tram is in standby, waiting for a client to make a request. This state is marked because it is assumed that if both vehicles are in this state, previous clients have already been serviced.

- Ti_w: In this state, a client has made a request, but neither vehicle has sent an election message yet.

[5]Having only two trams it is not strictly necessary that the tram send an election message after comparison, but a tram does not know how many trams are participating on the election

- Ti_d: The tram enters this state when it has received an election message from the other tram. It corresponds to a "processing" state, within which the distance from the other tram - received in the election message - is compared with the distance from tram i. The vehicle with smaller distance will be considered the winner, triggering a corresponding event.

- Ti_e: In this state the vehicle is in an election state, but not comparing distances. It waits for possible election messages to be delivered, or response(s) from the processing of the election message that it has previously sent.

- Ti_o: The vehicle is in this state when it is servicing the client. It reaches this state when it has won the election.

The alphabet of a tram in this scenario is composed of 8 events, of which 7 are mutual to both trams, and one is exclusive, the $\mathtt{d}$ event. Tram i contains only the event $\mathtt{d}_i$ between $\mathtt{d_1}$ and $\mathtt{d_2}$.[6] The alphabet of tram i can be defined as:

$$\Sigma_i = \{\mathtt{r}, \mathtt{e_1}, \mathtt{e_2}, \mathtt{s_1}, \mathtt{s_2}, \mathtt{w_1}, \mathtt{w_2}, \mathtt{d}_i\} \tag{3.4}$$

- $\mathtt{r}$: Same event as the one in the client model (section 3.4.3.1). Corresponds to a request being made, in which the client sends a message that is received by the vehicles.

- $\mathtt{e}_i$: Corresponds to the exchange of an election message. When $\mathtt{e}_i$ occurs, it signifies that tram i has sent an election message, and therefore that the other vehicle has received it and started processing it.

- $\mathtt{s}_i$: This event happens at the end of the processing of an election message. When a tram compares the two distances – the one received and the one it has –, one of the two will be selected as the vehicle to answer the request. If tram 1 has the smaller distance it will be selected and $\mathtt{s_1}$ will occur. Otherwise, if 2 has the smallest distance it will be selected and $\mathtt{s_2}$ will be fired.

[6] In the case of tram 1, the automaton only has $\mathtt{d_1}$ (and not $\mathtt{d_2}$). The opposite applies to tram 2.

The occurrence of an $\mathtt{s}_i$ event will involve, in the case of the tram processing the distances winning the contest itself, the exchange of an *election acknowledgment* message, in order to let the other tram know it has lost. Also the winning of the comparison means that the tram should also send an *election* message with its own distance value, because it cannot know if there are more trams participating in the election. If it loses, there is no message sent.

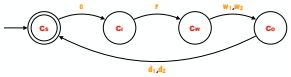
- $\mathtt{w}_i$: The same events already described in the client model, which correspond to the finalization of an election procedure. When $\mathtt{w}_i$ happens the vehicle i has won the election and will proceed to service the client. In this case it will send a message to the client and to the other tram, informing them of the occurrence.

- $\mathtt{d}_i$: As specified in the client model, this event occurs when the client is delivered at its destination, so that if $\mathtt{d}_i$ occurs the client has been delivered by tram i.

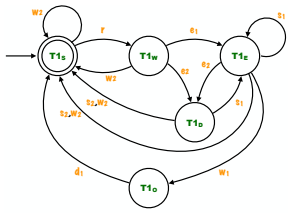
3.4.3.3 Global model

The system just described above is defined by the interactions between the two trams and the client. In the general case there could be any number of clients and trams. Therefore the model of the system as a whole is the union of client interaction and trams automata working together in a synchronized way.

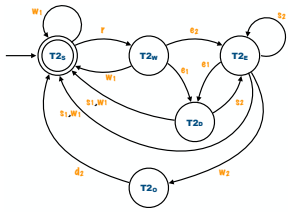
This conjunction of automata is carried out by the use of the parallel composition, previously defined in section 3.4.2. In order to obtain the automaton that models the whole system, one needs to combine the three other automata, obtaining the diagram observed in figure 3.10. This composition considers that any common events between the automata are synchronized – and thus can only occur if all automata with the event in their alphabet have that event active for their current states, that is to say, for a given event e and the automata in the system G_i: $e \in \Gamma_i(x_i),\ \forall G_i$ – and any private events do not affect the other automata that do not contain it in their alphabet, being allowed to happen at any time when they are active in their specific automaton.



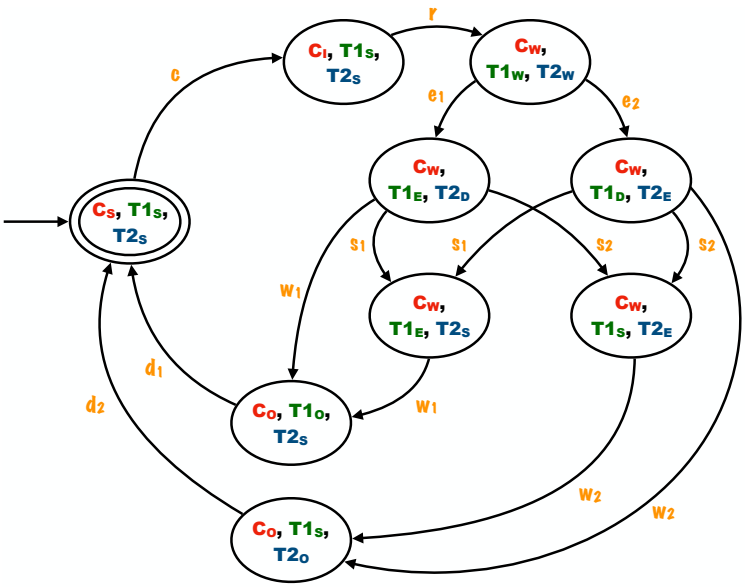
(a) Client automaton (C)



(b) First vehicle automaton (T_1)



(c) Second vehicle automaton (T_2)



(d) Global automaton (G)

Figure 3.10: System automata model for the case of two trams and one client. In the left are represented the client (a) and tram (b) and (c) automata, previously presented in sections (3.4.3.1 and 3.4.3.2). The global automaton (d) represents the combined behavior of all the components of the system. It is obtained through the parallel composition of the other three automata – $G = C \parallel T_1 \parallel T_2$ – and expresses all the possible combinations of the partial automata states in the system.

The resulting automaton, 3.10(d), shows that the proposed algorithm works, for it verifies that:

1. The client is always serviced

2. Moreover, it is serviced by one tram only

This is easily observed when taking into account the possible combination of events that the system can go through for each client that enters it. No matter what the sequence of events is, only one of the events $\mathtt{w_1}$ or $\mathtt{w_2}$ will occur, which means that only one tram will win the election procedure and consequently service the client. Also, the flow of events shows that in all cases, one of the $\mathtt{d}_i$ will always occur, and the system will always go back to the initial state (C_s, $T1_s$, $T2_s$), which means that the client has been delivered to its destination and the service has been

completed. There are no deadlocks in the system.

These conclusions are more easily observed in figure 3.11, which redraws figure 3.10(d) in linear format, to facilitate understanding. In this image, one can clearly see that the flow of events can only lead to one conclusion: the client will be eventually picked up and dropped off by tram 1 or tram 2.

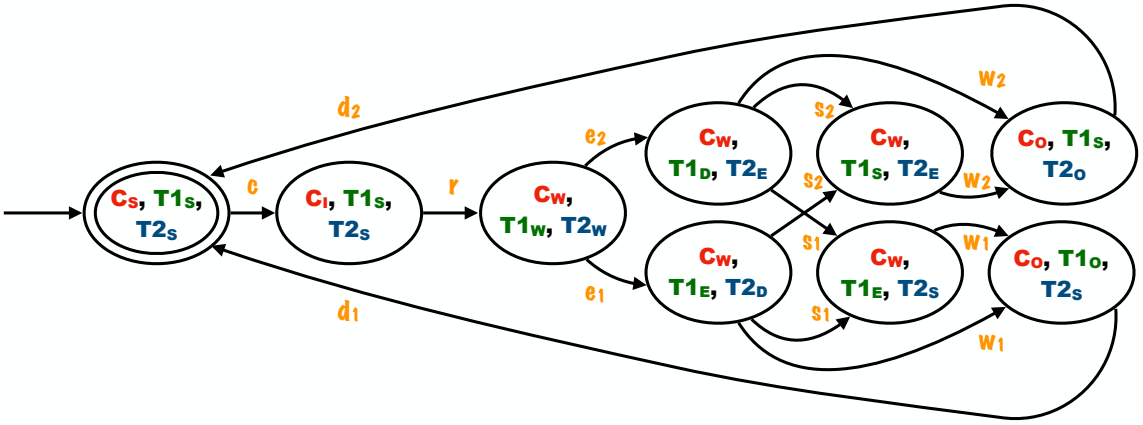


Figure 3.11: Global system automaton model, for two trams and one client at a time, reorganized into a linear flow of events (original: 3.10(d)). All events, except for the ones to drop off the clients, flow from left to right, and the system always goes back to the initial state, with no clients currently in the system and both trams in wait mode. For each client the system goes through client appearance, service request, election procedure and servicing the client.

Analysis of the language generated by the global automaton

In terms of languages, it can be said that the desired outcome of the marked language (in this case, the set of strings for which the system ends in the initial state itself) is that any string generated must always finish with the $\mathtt{w}_i\mathtt{d}_i$ substring, and that no other $\mathtt{w}_j$ or $\mathtt{d}_j$ can occur for the same client (even if $j = i$, for the election can only be won once).

To simplify the analysis of the generated marked language, only the result pertaining to one client cycle will be analyzed. One must bear in mind, though, that the full language can concatenate any number of such strings, and also contains the ϵ string, which means that no client ever appeared. The following are all the possible strings for the completion of one loop:

$$L_{\text{loop}} = \left\{ \begin{array}{l} \mathtt{cre_1 w_1 d_1}, \\ \mathtt{cre_1 s_1 w_1 d_1}, \\ \mathtt{cre_1 s_2 w_2 d_2}, \\ \mathtt{cre_2 s_1 w_1 d_1}, \\ \mathtt{cre_2 s_2 w_2 d_2}, \\ \mathtt{cre_2 w_2 d_2} \end{array} \right\} \tag{3.5}$$

The analysis of the given strings shows that, for all possible sequence of events for a given client appearance, the loops will always end with either $\mathtt{w_1 d_1}$ or $\mathtt{w_2 d_2}$, and no other $\mathtt{w}_i$ or $\mathtt{d}_i$ events occur, confirming the previous conclusion that all clients are serviced by one tram exactly.

3.4.3.4 Model extrapolation

The model developed here outlines the behavior of a simplified version of the proposed system. In this section, generalizations will be discussed, analyzing each generalization with respect to the modeled scenario. The objective is to ascertain what can be guaranteed in the more general cases, given that the simple case is guaranteed to work, and what are the possible points of failure of the system.

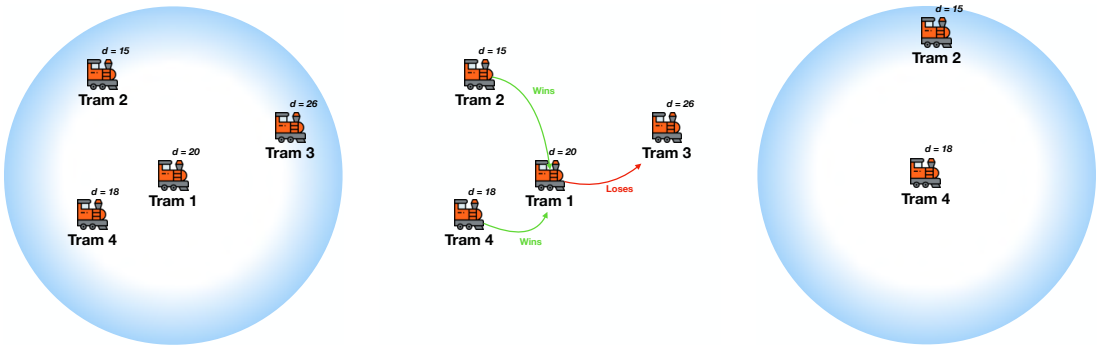
Different amount of trams

In the model it was assumed that there were only two trams in the system. However, for the actual implementation, this is an oversimplification. The system can have any number of trams operational at a given point in time, depending on the demand, general costs associated, project goals, etc.

The algorithm proposed, as explained in section 3.3, works by comparison of pairs of distances. So when a tram i sends an election message (triggering an $\mathtt{e}_i$ event), all trams that receive it will compare their own distances with the distance received. When they make such comparisons, for all intents and purposes, for each tram it is as if there are only two trams in the election: itself and the one which sent

the message. It will make a decision according to the possible scenarios depicted in figure 3.11 and equation 3.5: the receiving tram will either win that comparison, because its distance was smaller, or it will lose the election, which can happen because its distance was larger or because some other tram finalized the election before it had the time to finish its processing. Even if a tram receives more than one election message, they will be treated sequentially by each tram, and the conclusions still apply.

If no tram wins with respect to the one that sent the message in the first place, the latter will win the election and the procedure terminates. However whenever a tram wins a comparison round, it sends its own election message, making all other trams still within the election compare their distances with the new distance received. This is essentially equivalent to going back to the beginning of the election procedure, now with less trams.[7] Such behavior can be observed in figure 3.12.



(a) 4 trams available in the system, and tram 1 sends election message

(b) Trams compare the distance received with their own distances

(c) Tram 4 sends a new election message, since it won the comparison with tram 1

Figure 3.12: The images show the steps of an election with four trams. In image (a) the first tram sends an election message (represented by the blue circle), reaching the other three trams. They in turn compare their distances with the one received, arriving at the conclusions shown in figure (b). Trams 2 and 4 win from tram 1, taking it out of the election, and tram 3 loses, taking itself out of the election. Figure (c) shows the continuation of the election algorithm, in which one of the trams will send themselves the election value, in this case tram 4, beginning anew the distance comparison phase, now with two trams only, the case already guaranteed to work. Tram 2 is also expected to send its own election message, which is not illustrated here for greater clarity, assuming that tram 4 was faster and sent its message before.

[7] The first tram that sent the message has been taken out of the election, because some tram has won over it, but other trams may have lost against it and been taken out as well. The next step of the election, triggered by a new election message, can have several trams less than the previous.

In essence, the addition of more trams should not cause the breakdown of the decision algorithm, even if it makes its analysis more complex, as the automata would need to have more states and events (figure 3.13). In the same vein, having only one tram will not be detrimental, even if using the algorithm in such case is superfluous and just adds to the complexity of the problem; the only difference is that the tram will always win the elections, since no tram would ever answer with a smaller distance. The case of zero trams, however, would be a point of failure of the system. No clients would ever be serviced in such a case.

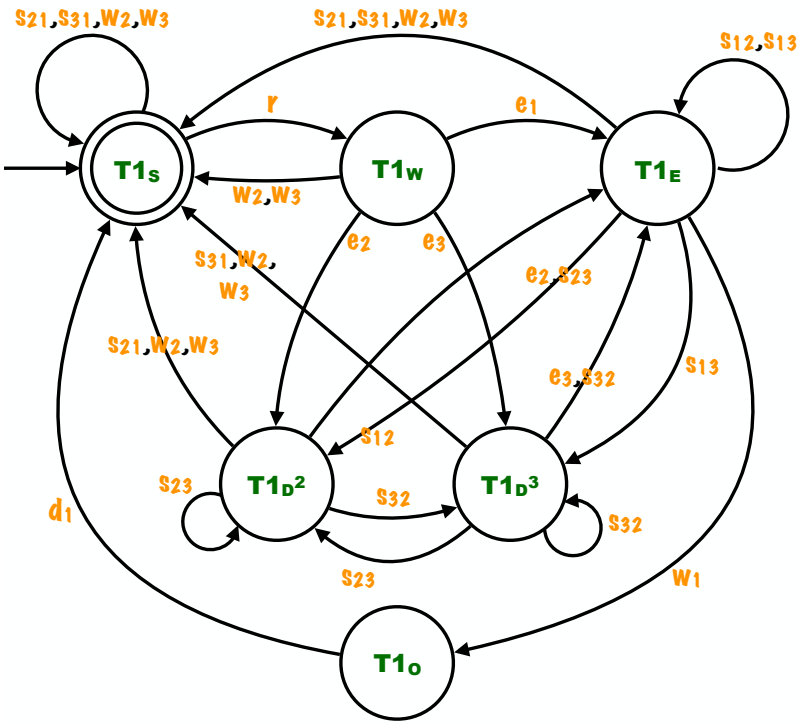


Figure 3.13: Tram automaton model (for tram number 1), considering the scenario of three vehicles and only one client at a time. The automaton behavior is similar to that presented in figure 3.9, but one needs to separate the comparison state into two, one to compare with each other tram. It is also needed to add events: e_3, w_3 and d_3, in the likeness of the existing events, and also modified s events in the form of s_{ij}, which stands for tram i winning the comparison with tram j.

Although it can be odd to specify the no trams scenario, because there is no sense in building a transportation system with no transportation unit, and moreover it will obviously fail, it is important to consider it, because as more and more generalizations are added to the model so as to make it more realistic, this will begin to be a plausible situation. For example, if one considers the possibility of

trams breaking down and stopping work for some reason, clients could be left with no trams left to service them. Another possible scenario is one in which the broadcast radius is reduced too much for the distances contemplated in the implemented grid. In such a case, further studied later, it could happen that there is no tram within reach of the client.

More than one client

We have previously assumed that only one client can be in the system at any instant in time. This is not a reasonable assumption for the global system, since one cannot limit the client demand in such a way. In the general case, there can be any number of clients at a time.[8]

Although the global system would be altered with the entrance of more clients, one thing remains the same: a tram can only participate in one election process at a time. This is a feature of the system design, chosen in order to simplify the election procedure.

In a manner similar to the reduction of the multiple trams problem into several two-trams problem, one can simplify the n clients problem into n one-client problems. For each client, we can separate the trams that are answering a specific client into a one-client-n-trams subsystem. Each subsystem is equivalent to the situation seen in figure 3.12, and thus guaranteed to function correctly. This behavior can be seen in figure 3.14.

Note that the figure shows a situation not yet contemplated by the extrapolations mentioned. It shows a situation where the request does not reach the whole system area, because this makes it is easier to understand the separation of trams into client-centered subsystems. However, even with the broadcast range covering the whole map area, some trams would enter or not a particular client election depending on their availability to do so. A tram would be unavailable to enter an election procedure if it was already participating in another election.

[8] However, in the case of trams usually the system design would stipulate previously how many trams the system would have available. For clients, the nature of the system itself stipulates that one cannot know exactly how many clients will appear, neither where nor when they appear.

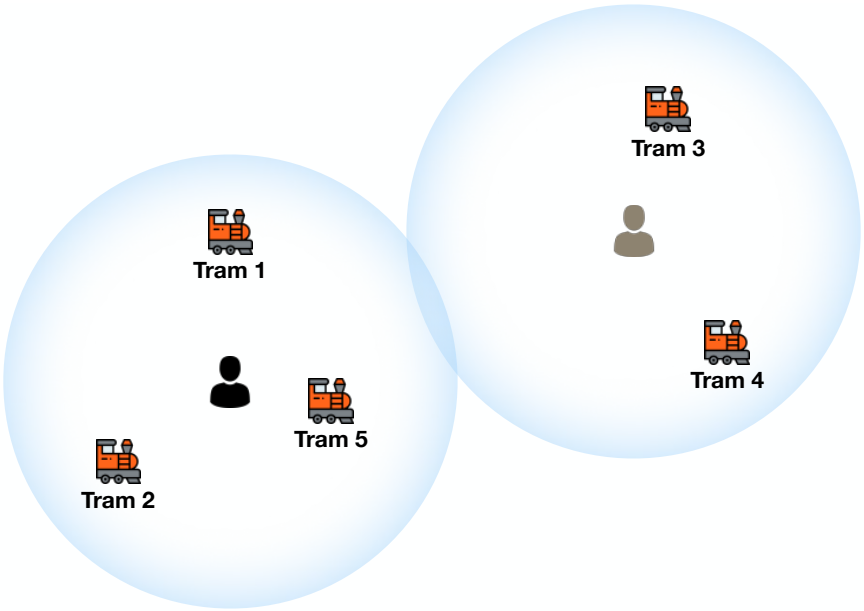


Figure 3.14: Illustration of the multiple clients extrapolation. The reach of each request is represented by the blue circles, and one can see that for each client there will be an arbitrary amount of trams that will work on that request independently. Note that if there was an overlap between the reach of the requests, the client that the tram would answer would depend on the timing of the requests, by order of arrival.

Although the system works with the tram automata modeled as they are in section 3.4.3.2, with more clients it would be a better description of the system if the Ti_o states were dropped. These states represent the actual picking up and dropping off of the clients by the trams, and this removal enables trams to participate in elections even while servicing clients, which would be the expected behavior. The modified tram automata can be seen in figure 3.15.

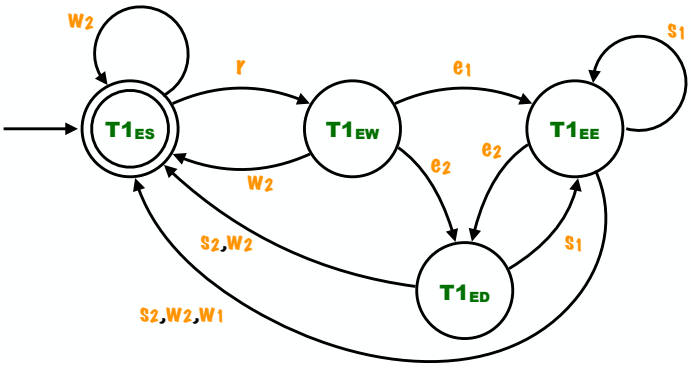


Figure 3.15: Tram behavior automaton (based on figure 3.9) modeling only the election process while disregarding the actual servicing of the client. The full tram behavior model would need to combine this automaton with another one, that describes the operational state of the tram (servicing or not servicing clients).

Broadcast radius

So far the underlying assumption has been that the client broadcast radius covers the full map area, which is implausible and probably unimplementable. Thus the system is designed so that this is not necessary.

If the client broadcast radius is too small it could happen that a client is unable to reach any tram. When the system is very busy, with trams moving throughout, the probability of this happening diminishes.

As a countermeasure to this problem, the system is designed so that trams answer a client to let them know their request is being processed, and if clients don't get this answer within an specified time interval, they resend their request. Although this does not guarantee that a client can never be without a tram, it decreases the probability significantly, specially when it is expected that the trams are always moving in the area.

Message delay

So far it has been assumed that message exchange is ideal, i.e., no messages are lost, nor is their order swapped, and furthermore they are assumed to be transmitted *instantaneously*. While this work will not consider the case of messages not delivered or having the receiving order scrambled, it will take into account that messages can have delays.

Messages can take long to be delivered, or even to be processed by the receiving agent. Keeping in mind that one cannot receive an answer to a message instantaneously, the election algorithm has a timeout to wait for the answers, and if this time interval is exceeded it assumes that no tram has a smaller distance than its own and supposes that it has won the election, prompting the sending of the *leader* message. In theory, the timeout should be large enough to give time for other trams to process and answer the election message, but even if it is not, in the ideal case represented, the receipt of the *leader* message immediately informs the other trams that they have lost, and should drop out of the election.

If messages can be delayed, however, it could happen that a tram wins the

election before it has the time to receive answer from some other tram, but this other tram does not receive the *leader* message until after it has already timed out and considered itself the winner. In this case, a single client would be serviced by two trams.

Although this is not as serious a failure as the case of a client not being serviced, from the point of view of the election algorithm that is the worst possible scenario: there is no consensus in the answer. The trams disagree on the algorithm outcome. To avoid that one needs to choose the timeout value carefully, taking into account, for example, how busy the system is, because a greater number of elections taking place implies an increase in message exchange, which in turn increases the response time.

Chapter 4

Simulation description

The proposed transportation system was implemented in the form of a computer simulation. The source code is available in the GitHub© repository V.A. [7] from user *AmieOliveira*. In this chapter the implementation details will be elaborated on. The operation options and decisions will be specified, code flow diagrams presented, as well as specifics of the simulation created.

4.1 Request handling modes

As thoroughly explained in section 3.3, when a client request is made, the trams will communicate and decide which one will handle the request through the election algorithm. However, for the algorithm to work, each tram needs to calculate their distance with relation to the client, which will be in turn used as a comparison parameter.

Although the distance associated with the election winner aims to minimize the client total service time (defined in section 2.2), the distance value will depend on the mode of operation implemented in the system. For the first mode, in the *v3* code versions, the trams only pick up one client at a time. That means that one client needs to be delivered before a tram can go to pick up the next one. On the other hand, the *v4* code versions introduce a *ride sharing* possibility, in which two or more clients can be serviced by the same tram at the same time. These modes,

and the distance calculation in each case, will be detailed below.

4.1.1 Single client mode

In this version, implemented in code version *v3*[1], trams can only transport one client at a time, even if they have several clients in line to be delivered.

Since there is only one client at a time, any tram will always choose the same path to deliver the client from its origin to the destination. Thus in order to know which tram will deliver the client faster, one only needs to compare the distance the trams will take to get to the client origin. This is the distance used by the trams in the election algorithm.

4.1.2 Multiple clients mode

Version v4 of the code[2], introduces the possibility of sharing rides between clients. That means that a tram can pick up a new client before it finishes the process associated with its original client.

In this work, only one possibility of sharing is contemplated: the case in which the tram does not need to change its route in order pick up the client. So if the new client happens to be on the route determined in order to service the original client(s), ride sharing will occur and the tram will pick the client up before heading to deliver the current client(s). The clients will be delivered in order of pick up, unless the drop-off location of the new client in on the route to the delivery of the original one(s). That means that, even if the tram picks up a client on the way, the client that was already scheduled to be delivered will be delivered first, and only then will the tram head to the second client destination. The route to service the original clients is not altered by the addition of a new one.

With this modification, the previous distance calculation no longer holds. In order to guarantee that the total service time will be minimized, one now needs to

[1] This version is, as of December 2019, implemented in the *master* branch of the project.
[2] As of December 2019, this version was implemented in *multipleClients* branch.

consider the full distance until the client is delivered instead of the distance until the client origin. This implies that the tram chosen is not necessarily the one that reaches the client faster, but the one that will deliver it to its destination faster. This situation is exemplified in figure 4.1, in which, if the same distance calculation in the single client mode were used, tram 0 would be the one selected to service client 0, when it would in fact take longer to deliver the client to its destination than tram 1.

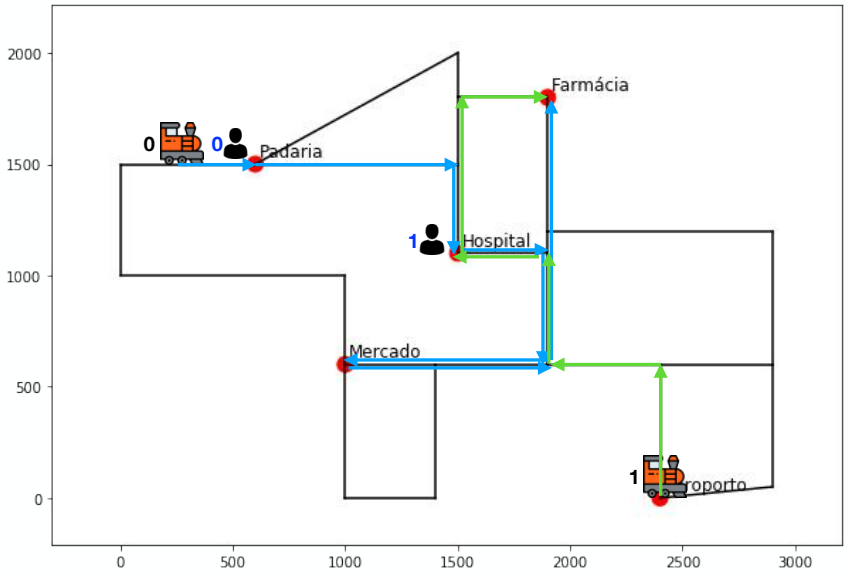


Figure 4.1: Consider the situation, in which the code runs in multiple clients mode, client 0 is going to *Mercado*, and is scheduled to be picked up by tram 0, and client 1 wishes to go to *Farmácia*.
In the given situation, tram 0 is the closest to client 1, with a distance of 1.5km to the client position while tram 2 is 2km away from the client. However, since tram 0 is first picking up client 0, the actual distance it would have to go through to deliver client 1 is of 5.5km, while tram 1, that is not delivering any client beforehand, manages to deliver client 1 in 3.2km.
Therefore, in this context, tram 1 should be the one selected by the election algorithm, even if tram 0 would manage to reach the client position faster.

4.2 Simulation organization

The simulation library is subdivided in five main files, each coordinating the behavior of one of the main aspects of the system. The first four are part of the code library and coordinate the tram behavior, the client behavior, the message protocol and the

broadcasting behavior of the message exchange. Besides those, there is a *Simulation* file, which organizes all other aspects and runs the system simulation itself, also providing a visual interface of the system.

The broadcasting behavior is modeled as a *network* object that delivers the messages to all agents within the message broadcast radius, according to the behavior described in 3.2. On the other hand, the message protocol serves the purpose of encapsulating messages in an established format, creating a pattern that all agents are capable of correctly interpreting, allowing flawless communication. These two files enable the message exchange in the system.

The client behavior follows the flow diagram in figure 4.2. The client recursively passes through the stages of updating timers, handling received messages and, if that is the case, sending messages.

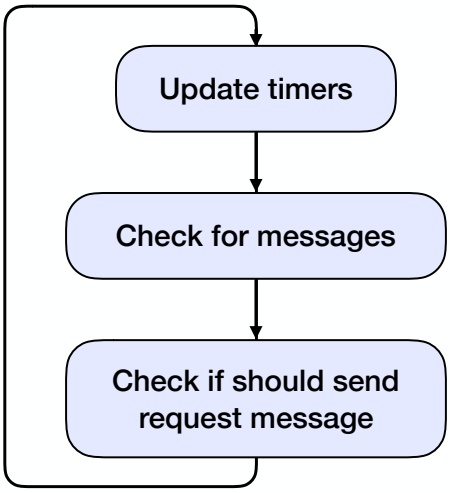


Figure 4.2: This is the flow of actions taken by the client at each simulation loop. It begins by updating its timers, then checks if any messages were received, taking necessary actions in the case there was any, and then, in the case it has not yet sent the request or not gotten an answer from any tram for too long, it broadcasts an election message.

Any incoming messages need to be adequately processed by the client message handling protocol. Clients need to be able to discard messages that are not meant for them, since the broadcast message exchange implemented means that all messages reach all agents. Furthermore, clients need to respond accordingly to messages that are meant for themselves.

There are four types of message that a client can receive, which are listed and explained below.

- *req_ack*: An answer message from the tram that lets the client know it has received and will process its request. This message signals to the client that it does not need to send any more request messages, since the message has already been received.

- *req_ans*: The client receives this message when the election procedure is finished and the tram has been assigned to service it. It registers the tram identification number and awaits arrival.

- *pickup*: Sent by the tram when it has arrived at the client location, to let it know it should board the tram.

- *dropoff*: Sent by the tram when it has arrived at the client destination, to let it know it should leave the tram. This is not strictly necessary for actual system implementation, but an action that would have similar effect would be for the tram to have an audio or visual notice inside reporting arrival at the station.

Although other types of messages that could reach the client exist, they will be automatically discarded, for their meaning is not included in the client application interpreter. Furthermore, in the simulation the client is only capable of sending the request message (*req*), to request transportation between stations. The possibility of changing their mind and canceling a request is not contemplated, as specified in section 3.3.

The tram behavior follows the diagram shown in figure 4.3. Like the client, the tram recursively passes through the stages of updating timers and handling pending messages, but then their actions diverge, for the tram will verify next whether it should begin or end an election, and finishes by updating its speed and position.

Trams also need a message handling protocol, which will operate differently from

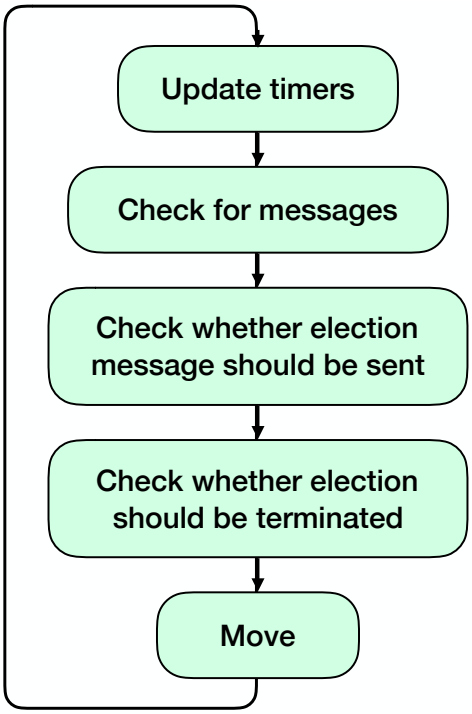


Figure 4.3: This is the flow of actions taken by the tram at each simulation loop. It begins by updating its timers, then checks if any messages were received, and if so, then they are properly with according to the message type. The tram then will either begin or end an election procedure, if the current state of operations enables it, and lastly the tram will update its speed and position in the map.

the clients one. For any message received, the tram first checks that it pertains to itself and if so takes necessary action, according to the type and content.

Trams may receive four types of message, all of which have previously been covered in the election algorithm design in section 3.3. In the code, the request message is called *req*, the election message *elec*, the election acknowledgment *elec_ack* and the leader message suffers no change, still being called *leader*. The trams can send all types of message so far mentioned, except for the *req* message, that pertains only to clients.

4.2.1 Tram movement

The trams are assumed to move at a constant speed at any point in the map, with the exception of stops at stations or intersections. This speed is an immutable simulation parameter, common to all trams and is referred to as maximum tram

speed.Furthermore, as specified in section 3.1, the trams are capable of calculating their routes using the Dijkstra Algorithm, which was implemented using the *NetworkX* library [13].

In the simulation the map of stations was considered fixed. Since there are no changeable paths and no traffic congestion, whenever a tram goes from one specific station to another, it always takes the same path. For this reason, in order to diminish the computational load of trams, their code saves the paths from one station to another, for all pairs of stations. This way, it only needs to really calculate the shortest path through the Dijkstra Algorithm once for each possible path, and will only go through the saved database during its operation to retrieve the desired pathways.

However, if during operations the tram finds itself in the position of having to take a way not in the database (if its initial and/or final position is not a station), it will use the Dijkstra Algorithm in real time to acquire the required path.

Collision avoidance

As specified in section 2.1.1, besides being able to calculate routes and move independently, trams need to be able to avoid collisions between themselves. It was also assumed that there is an extra platform area (of infinite capacity) for trams to wait between intersections, so that one can vacate a track for the other without collision. When in an extra platform, vehicles do not occupy any track.

In this work this is implemented with the use of semaphores. Each track can only have one tram traversing it at a time, and the semaphore will signal to the trams whether the track is vacant or not. These semaphores are local, each track should have its own independent semaphore, and these should communicate with the trams so that it shows busy when the track is already in use, or vacant when not. If a tram enters the track its semaphore should immediately turn to busy without letting any other tram enter as well, and when the trams leave the track, the semaphore should immediately be set to vacant again. The tram that waits for the track to be available again should do so at the extra platform in the intersection between

tracks, having freed the previous track it occupied and allowing trams to leave the adjacent tracks unimpeded.

The use of semaphores can cause a *race condition* between multiple trams that are waiting to use a given track. A race condition is a situation where the system behavior depends on the sequence and/or timing of uncontrollable events. In this case, the race condition applies to the order of usage of the track by the trams.

For this work, it was considered that the order of usage is not of much importance, for all trams will eventually have access to the track. The essential point is to disallow simultaneous access to a track by more than one tram, since this will lead to collisions. In the simulation code, this is guaranteed not to happen due to the sequential nature of the simulation. In other words, when a tram is checking a semaphore, it is always the only one doing so, and its code will always lock the semaphore into busy when it enters the track before any other tram has the chance to also check the availability of the track.

In a real situation, each track control system would need to have the means of guaranteeing that only one tram can access and modify the semaphore at any point in time. This can be achieved by the supposition that all requests for a certain track are processed by a centralized control. System scalability can be maintained by keeping all track controls independent from each other (configuring a decentralized access to tracks).

4.2.2 Client demand modeling

When run in standalone[3], the simulation generates a random client demand in real time, applying a normal distribution in time and space. This means that any station has the same chance of having a client appearing, and the same can be said for being chosen as destination. At the same time, all instants in time have the same chance of being a moment when a client decides to enter the system. This distributions are

[3] Running the simulation in standalone means using the *Simulation* file – mentioned in section 4.2 – to run the system. One can alternatively produce a variation of the code presented in this file to apply different system conditions.

made with the use of the *random* python library [14].

However the code can be run with any demand distribution, as long as one creates the desired dataset and applies it to the simulation. The dataset should be a set of three arrays: one to determine, for a given loop in time, whether a client will appear, and two to determine the origin and destination stations of the client, in the case the later is created.

In order to determine whether a client will enter the system, one has a number between 1 and 100 for each simulation loop, and this number is divided by the code parameter named *client arrival frequency parameter* (f_c). If the reminder of the division is zero, that is, if the number obtained is a multiple of the parameter, then a client is generated. This frequency parameter is further detailed in section 4.3. The stations are chosen by assigning sequential integer numbers to it, and then one amongst these numbers is raffled as the origin, and another as the destination. So, for example, if there are 5 stations, they would each be represented by a number between 0 and 4, and a client could wish to go from 3 to 0, or any other combination, for that matter.

4.2.3 Simulation interface

The simulation prints an animation, done using the *matplotlib* library [15], of the system behavior throughout its operation. This shows the map of the track grid, with the trams and clients in it. A screenshot of the interface can be seen in figure 4.4.

Besides the trams and clients, the interface also has a timer on the bottom right side. This represents the time in human units of hours, minutes and seconds. Furthermore there are two interactive buttons: *Play/Pause* and *Stats*. *Play/Pause* allows the user to pause the simulation mid-operation, which can be useful when there are many agents interacting and one needs time to process what is happening in the system. The *Stats* button shows an extra window with some statistics of the simulation, specifically the number of already delivered clients, the average total

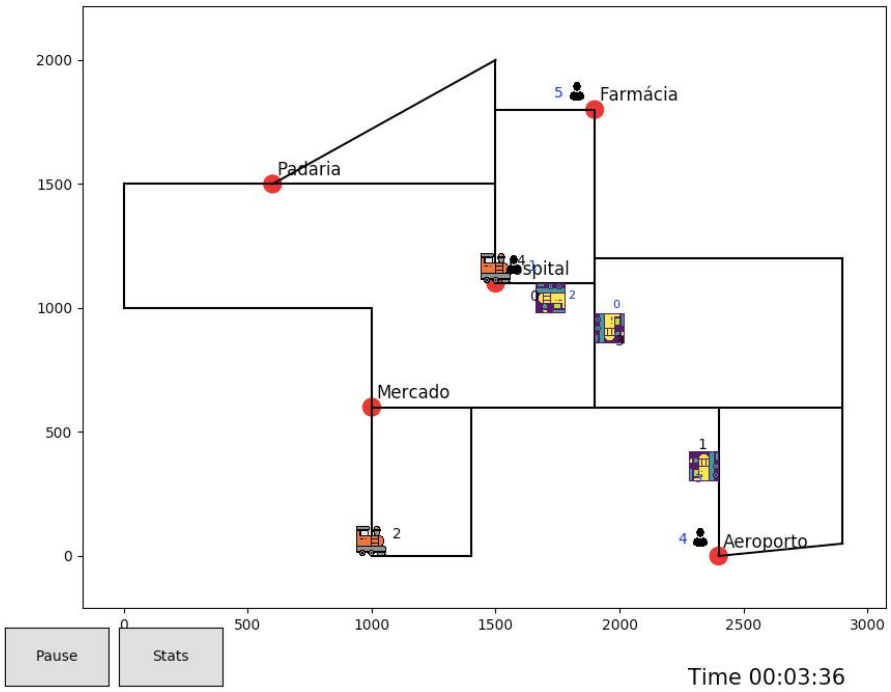


Figure 4.4: The image shows a snapshot of the simulation interface. In it, one can see the clients waiting at the stations, represented by red dots and names, and the trams moving around servicing the requests. The trams and clients identification numbers can be seen beside them, the tram numbers in black and client numbers in blue. For example, tram number 1 has just left the *Aeroporto* at the moment this image was made. The trams in yellow are the ones that have clients on board, the clients identification is shown next to the tram in which it is traveling. This is the case of client 2, that is traveling in tram 0.

service time, and the total distance traversed by all trams. Figure 4.5 shows a screenshot.

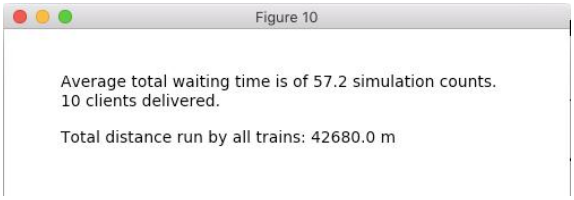


Figure 4.5: The image portrays the statistics window shown when the user presses the *Stats* button during the simulation. This box shows the number of already serviced clients, with the average total service time achieved, and also the total distance traversed by trams until that point in the simulation.

The interface also shows the broadcast range of any client request. When a client makes a request, a blue circle appears around it, showing the area this request reaches, and which trams should process this request, if they are available to do so. This is illustrated in figure 4.6.

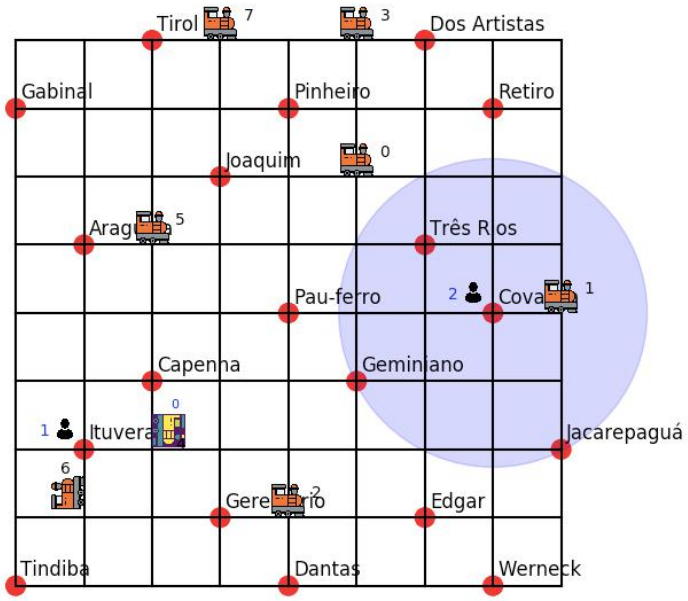


Figure 4.6: The image shows the simulation interface map when a client requests the service. In this case, client 2 was making a request, and its reach means that tram 1 received it.

4.3 Simulation parameters

There are several project variables and parameters that need to be specified in order for a simulation to be performed. Several conditions affect the system behavior, some of which will be fixed by the code implementation and others that can be changed according to user's needs and desires.

Simulation step speed

The simulation is organized in the form of a loop, and at each simulation loop, called a step, clients and trams update their states, as described in section 4.2. To convert the the number of simulation loops into human time, there is the step speed variable, measured in seconds per step. The default value of this variable is 1 s/step.

Ideally this value is selected according to the system processing capabilities. That is, this variable is chosen to reflect how fast the trams are able to transmit and process messages, so that the simulation functions as the real system would.

Election timeout

The election algorithm uses the timeout parameter in a crucial manner. It is the time a tram waits for answers from a sent election message. This parameter is key for the algorithm to work, for if it is too small, it will lead to lack of consensus, and if too big will lead to unnecessary wait. This parameter will be further explored in chapter 5.

Broadcast radius

In section 3.2, the message exchange behavior was explained, in which agents communicate through broadcast. One critical aspect is the broadcast range, since it directly influences which trams will participate in the election. If the range is too small, it could happen that no tram receives a request, which would then remain unanswered, as discussed in section 3.4.3.4. At the same time, one does not want the broadcast radius to be too large, since this would cause a grater number of messages to be exchanged and also increase client waiting times.

The broadcast range or radius is specified as a fraction $f_d \in (0, 1]$ of the map diagonal d_{map}[4], i.e., the client broadcast radius is set to $r_c = f_d d_{\text{map}}$.

It is important to notice, however, that for the election algorithm to work, the tram broadcast radius cannot be the same as the client one, or trams processing the same request could be unable to communicate. The lower bound for the tram radius is twice the client radius, considering the situation where there are two trams located at diametrically opposed ends of the range circle. This situation is shown in figure 4.7.

Supposing that the trams are both stationary, choosing the tram radius be $r_t = 2r_c$ is enough to enable communication. However, since trams might be moving in opposite directions, away from each other, the margin of the tram radius over $2r_c$ should be more than the maximum distance traversed by trams during the election

[4] The map diagonal is the diagonal of the smallest rectangle that contains the whole tracks area.

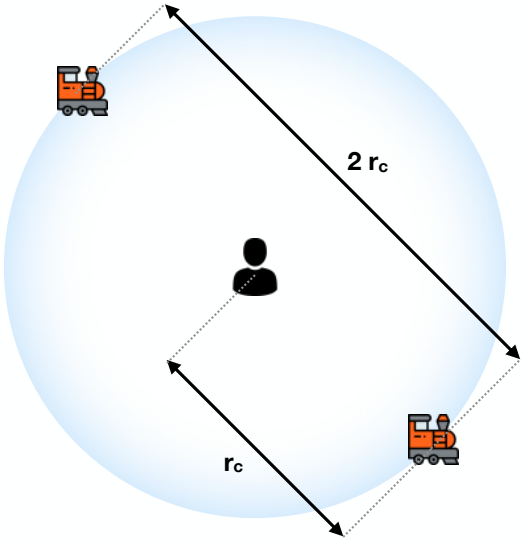


Figure 4.7: The figure shows an illustration of the client broadcast range, with two trams diametrically opposed to each other in the outermost circumference of the message reach. In such positions, both trams would receive the client service request message, and be expected to react accordingly. That means that both trams need to communicate, and therefore the tram broadcast range must necessarily be at least twice the client range, otherwise it is assured that in cases as this, communication will not be able to occur.

procedure. This is shown in equation 4.1:

$$r_t \geqslant 2r_c + 2V_{max}t_{max} \tag{4.1}$$

In this equation, V_{max} is the maximum tram speed, and t_{max} is the maximum time interval the election should last. This distance $2V_{max}t_{max}$ is calculated considering the worst case scenario: both trams are moving away from each other at full speed during the whole election time.

Number of trams

The simulation can be run with any desired amount of trams, one only needs to create the tram objects. In standalone mode, with the *Simulation* file, the number of trams is an argument chosen at runtime and trams are created in random positions at the map.

Client arrival frequency

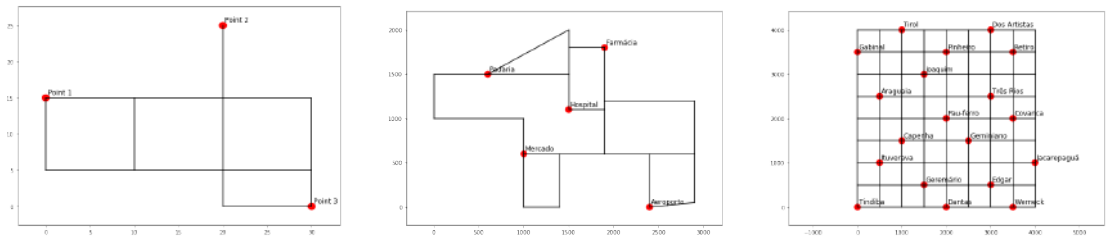
As discussed in section 4.2.2, the simulation is normally run in standalone mode, with the *Simulation* file. In this case the client appearance will follow a uniform distribution, and the rate of appearance in time will be given by the *client arrival frequency parameter* f_c.

In each simulation loop, a random number between 1 and 100 is generated. If this number is divisible by f_c, then a client appears. For example, if $f_c = 20$, a client will appear if the number generated in the loop equals 20, 40, 60, 80 or 100.

If one alters the simulation code and uses a previously generated data set, then any distribution can be applied, with or without the f_c parameter. In the tests here performed, however, this parameter was still used, even when the tests used a dataset.

System map

In order to run the simulation, the system needs to have a graph of tracks and stations previously defined. The map is given to the simulation as an input in the form o CSV files, which are in turn interpreted by the code. The library folder has three map examples, which are portrayed in figure 4.8.



(a) Basic Map, available as *mapFile* in the library folder

(b) Dog Map, available as *mapBig* in the library folder

(c) GRID Map, available as *map_grid* in the library folder

Figure 4.8: Three examples of possible railway graphs

Boarding time

It is considered that clients need time to board and to vacate trams, and this time is implemented in the simulation. When a tram arrives at a the client position, it

waits a specified time for it to board. In a similar fashion, when arriving at the client destination, the tram waits to give the client time do disembark.

Chapter 5

Experiments

This chapter contains a series of experiments performed, using the simulation code developed, and designed to show the system behavior with respect to variations in system conditions, as well as its critical points and observed failure cases. All the experiments are registered in iPython notebooks in the GitHub folder.

The first series of experiments demonstrates that the system works correctly, according to specifications. The next set of experiments evaluates system performance.

5.1 Verifying correct operation of the code

The first step in verifying correct operation is to run the code in standalone mode. The code can be run with a shell command, as described in the *README* file of the repository, and results in an animation of the simulation, in real time. A screen shot is shown in figure 5.1.

The animation shows the movements of trams and clients in the map, their behavior governed by the algorithms and specifications detailed in chapters 3 and 4. A brief glimpse of what is executing behind the screen can be seen in the logs printed on terminal when the simulation is run.

However, accompanying a specific simulation and verifying that it behaves as expected does not validate the code, and, furthermore, becomes untenable for a large

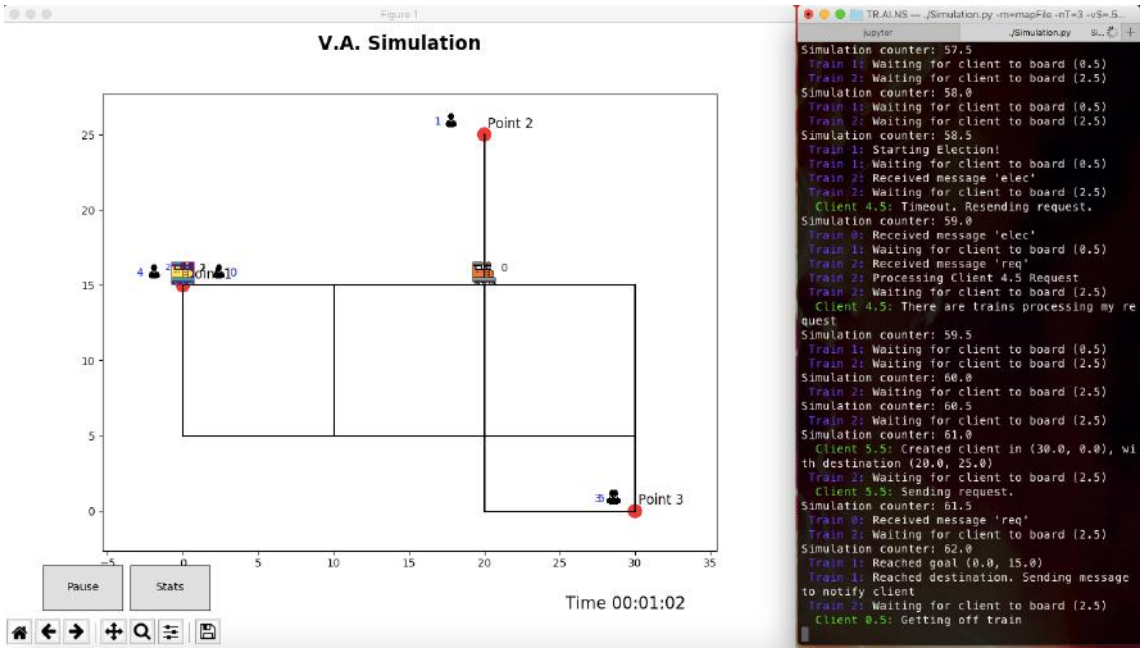


Figure 5.1: Simulation animation interface with the tram and clients logs on the terminal to the right.

number of clients and trams. In order to credibly assess the system operation, tests were performed in which 15 clients appeared at random stations in the map (and with random timing), with each test terminating when all 15 were duly delivered to their destinations. For each set of conditions, the simulation was repeated 60 times, in order to be able to compute statistics, and increase the possibility of detecting rarely occurring bugs. The initial positions of the trams are randomly generated for each simulation. Table 5.1 shows the parameters used in the experiment.

The tram speed, the simulation step speed and the boarding time are not going to be varied in the performed experiments, being always equivalent to 20 m/s and 1 s/step and 10 s. f_c (client arrival frequency parameter) is going to be varied only in one of the experiments, in section 5.2.3. Also the election timeout used will be of 45 s, with the exception of the experiment in section 5.2.5, which will explore this parameter further.

In this work, the client broadcast radius is chosen as a percentage of the map diagonal d_{map}, as explained in section 4.3. Since the GRID map has a map diagonal of approximately 5657 m, the client broadcast radius varies between 2263 m and 5657 m. For such conditions the choice of the tram broadcast radius is enough to

Client arrival dataset:	normal_map-grid_60rep_2000iter.npz
Map:	GRID map (figure 4.8(c))
f_c **parameter:**	20
Tram speed:	20 m/s
Simulation step speed:	1 s/step
Election timeout:	45 s
Boarding time:	10 s

Number of trams:	{5, 6, 7,8, 9, 10, 11, 12, 13, 14, 15}
Client broadcast radius:	$f_d d_{\text{map}}$, $f_d \in \{40, 50, 60, 70, 80, 90, 100\}$
Tram radius:	3 times client radius

Table 5.1: Parameters used in the experiments to verify that the code operates correctly. The first set of parameters are the conditions shared by all simulations run. The number of trams and broadcast radius are varied through simulations and therefore are given as a set of values. For the GRID map, $d_{\text{map}} = 5656.85$.

satisfy inequality 4.1, considering the tram speed and a supposed maximum election duration of one minute:

$$r_t \geqslant 2r_c + 2 \times 20 \times 60 = 2r_c + 240 \tag{5.1}$$

The simulations performed were done for both the single client mode, and multiple client mode, explained in sections 4.1.1 and 4.1.2 respectively. The results for each case will be separately presented in the next sections.

5.1.1 Single client mode experiments

The results here presented were obtained with code version *v3.4*, i.e. no ride sharing. The experiment is registered in iPython notebook *"Simulation Test - Number of trains by broadcast radius - Preset dataset - Grid - v3.4 - 15 clientes.ipynb"*. Figures 5.2, 5.3 and 5.4, extracted from this notebook, represent the time parameters observed, while figure 5.5 shows the total distance traversed, which is the sum of the distances traversed by all trams during the simulation, as defined in section 2.2, as a function of the number of trams in operation.

All the 4200 simulations – 10 different number of trams, 7 different broadcast radii, 60 repetitions for each case – performed as expected, with all 15 client requests

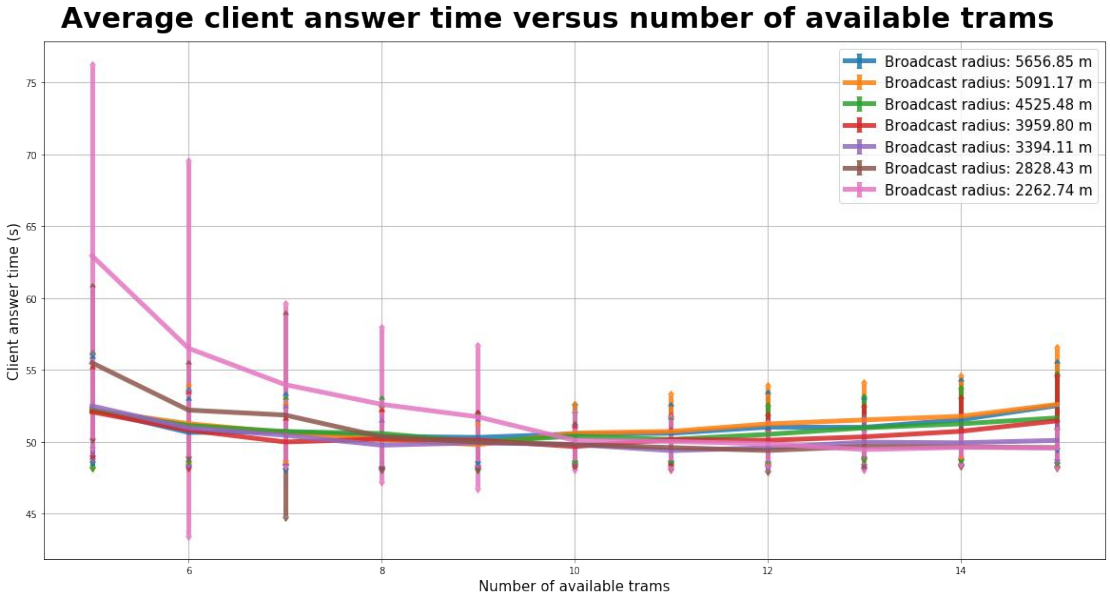


Figure 5.2: The image shows the average single client mode answer time for each number of trams and broadcast range combination, portraying the standard deviation of the measurement in the error bars. The Y-Axis represents the answer time in seconds, and the X-axis the amount of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 60 experiments were carried out.

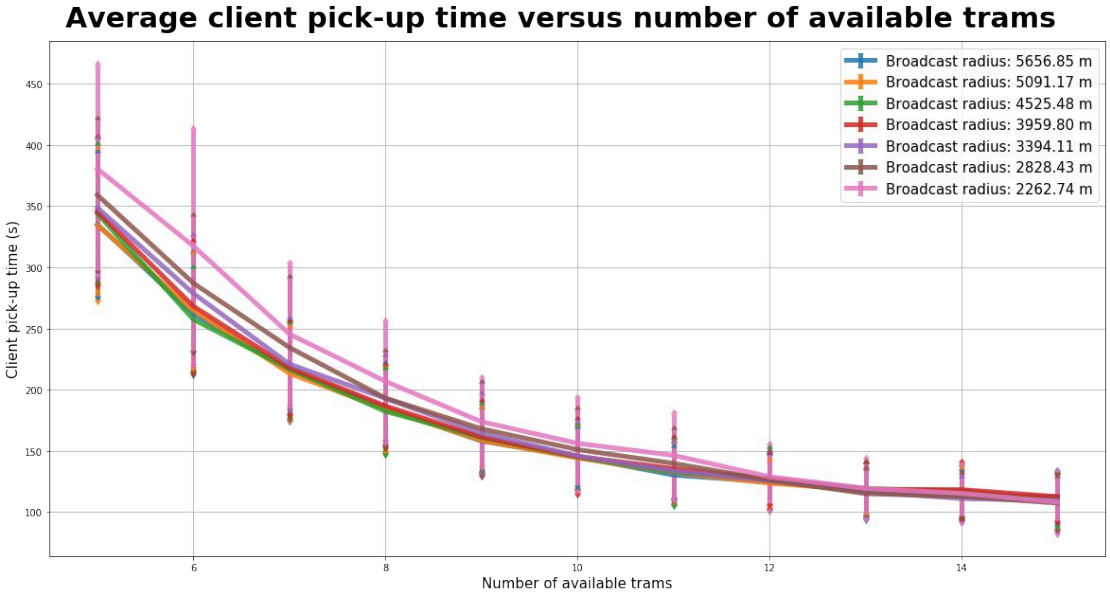


Figure 5.3: The image shows the average single client mode pick-up time for each number of trams and broadcast range combination, portraying the standard deviation of the measurement in the error bars. The Y-Axis represents the pick-up time in seconds, and the X-axis the amount of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 60 experiments were carried out.

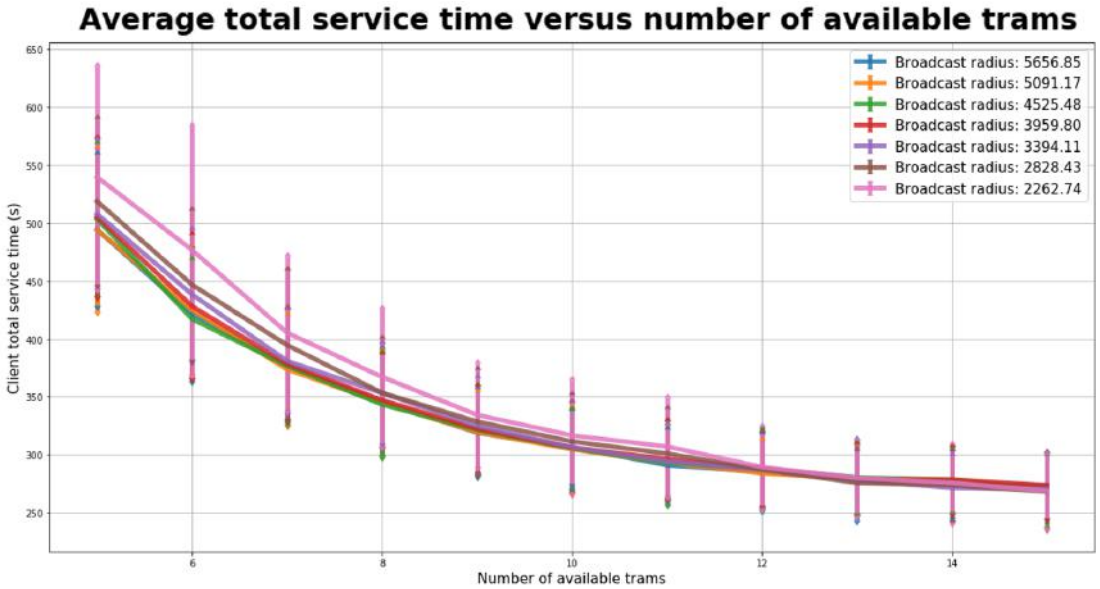


Figure 5.4: The image shows the average single client mode total service time for each number of trams and broadcast range combination, portraying the standard deviation of the measurement in the error bars. The Y-Axis represents the total service time in seconds, and the X-axis the amount of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 60 experiments were carried out.

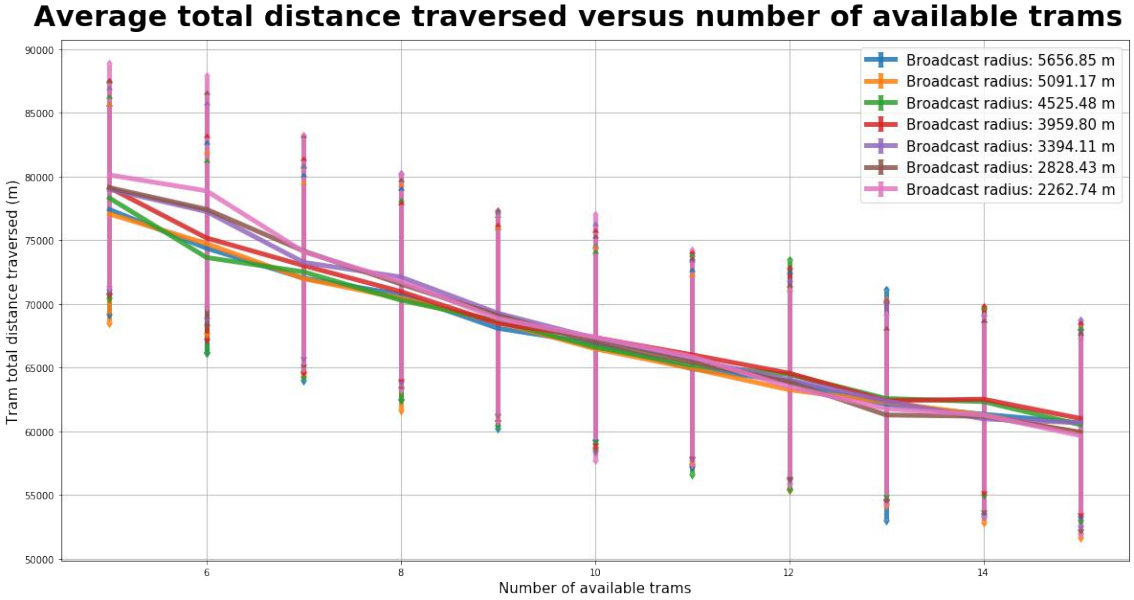


Figure 5.5: The image shows the average single client mode total distance traversed by all trams for each number of trams and broadcast range combination, portraying the standard deviation of the measurement in the error bars. The Y-Axis represents the distance in meters, and the X-axis the number of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 60 experiments were carried out.

being serviced and no errors being shown in the logs.

The graphs show several expected behaviors. Figures 5.2 through 5.5 show that the client broadcast radius does not interfere greatly in the obtained results. The graphs shown are very similar for the same parameter, with the exception of the light pink graphs, which correspond to a client broadcast radius of 2262.74 m, that is, 40% of the map diagonal, for the three time measurements. With less available trams, this smaller broadcast radius increases the average waiting times considered, especially the answer time. This occurs because a smaller broadcast radius increases the chance that there are no available trams close enough to the client to receive and subsequently answer the request.

Similar reasoning applies when the number of trams is diminished. This is the reason why the simulations begin with a broadcast radius of 40% of the map diagonal: below this value, failure to service a client could occur. If chosen smaller than this, the simulation was not guaranteed to work on all occasions.

With respect to the number of trams, it can be observed that the pick-up time and total service time diminish with the increase of available trams, which is expected, for more trams should be capable of organizing themselves better and service clients faster.

For the total distance traversed by trams, the graphs also show that an increase in the number of trams corresponds to a decrease in the distance traversed. However it should also be observed that the uncertainty of these measurements is very high when compared with the difference in the measured values themselves. So the relevance of these distance measurements must be questioned. The experiments suggest that the great relative uncertainties are due to the nature of the simulation itself: even restricting the system variables, the distance to be traversed by all trams has a very random nature, in the sense that it strongly depends on the clients and trams positions at all times, which can vary a lot from one simulation to the next.

The answer time, in figure 5.2, deviates from the pattern shown for the other measurements, for it is much less influenced by the amount of trams. This is ex-

pected, since the biggest influence factor should be the election timeout. However one can notice a trend, especially for the three biggest broadcast radius (in blue, yellow and green) of an increase in the answer time for a bigger number of trams. It is speculated that this is se because many trams in the system causes many more messages to occur, which delays the reading and processing of such messages, in turn increasing the answer time.

5.1.2 Multiple clients mode experiments

First, in order to guarantee that ride sharing occurs in this version, the experiment registered in iPython notebook *"Simulation Validation - Multiple clients version.ipynb"* was performed. In this, two trams were created, one starting in *Tindiba* station and another in *Geminiano*. Four clients were created:

- Client 0 appeared in the fourth simulation step at station *Ituverava*, requesting a service to *Três Rios*.
- Client 1 appeared in the 34th simulation step at station *Araguaia*, requesting a service to *Covanca*.
- Client 2 appeared in the 127th simulation step at station *Dantas*, requesting a service to *Joaquim*.
- Client 3 appeared in the 156th simulation step at station *Geremário*, requesting a service to *Edgar*.

These clients were specifically chosen in time and position to firstly enable ride sharing between clients 0 and 1 and, secondly, to create a situation between clients 2 and 3 that, even though ride sharing was possible, it should not be chosen because client 3 would take less time to be delivered if another tram picked it up, without ride sharing.

The results observed in this test show that, using the multiple clients mode, clients 0, 1 and 3 are picked up by tram 0, with ride sharing between 0 and 1, and

client 2 is picked up by tram 1. There was a decrease in all average time parameters, as well as the total distance traversed, when using version $v4.1$, i.e. with ride sharing, instead of version $v3.4$, that is, without ride sharing. The results can be seen in table 5.2.

	Client 0			**Client 1**		
	Answer	**Pick-up**	**Total**	**Answer**	**Pick-up**	**Total**
Single Client	28	67	175	28	106	201
Multiple Clients	28	67	180	24	80	180

	Client 2			**Client 3**		
	Answer	**Pick-up**	**Total**	**Answer**	**Pick-up**	**Total**
Single Client	28	148	244	28	176	219
Multiple Clients	27	79	174	25	160	203

	Total Distance Traversed
Single Client	24000
Multiple Clients	17000

Table 5.2: Table with the results of comparison between single and multiple clients modes. *Answer* denoted the answer time, *Pick-up* the pick-up time and *Total* the total service time. It can be observed that there is always less or equal answer and pick-up time. The total service time is less on average, though it happens to be higher in the case of client 0, which is attributed to the time waiting for client 1 to board.

Having tested that the ride sharing is functioning appropriately, the same conditions previously used for single client mode were applied to multiple clients mode, using code version *v4.1*. The experiment is observed in file *"Simulation Test - Number of trains by broadcast radius - Preset dataset - Grid - v4.1 - 15 clientes.ipynb"*.

In general, the same observations made for the single client made can be applied to the multiple clients mode results. It was also observed that all simulations functioned accordingly and performed as expected. The results observed are shown in figures 5.6, 5.7 and 5.8, for the waiting time parameters, and in 5.9, for the distance traversed parameter.

Overall, there are two important observations that can be made about the graphs portrayed, for both the multiple clients mode and the single client mode. Firstly, that the higher amount of trams diminishes the total service time for clients, which

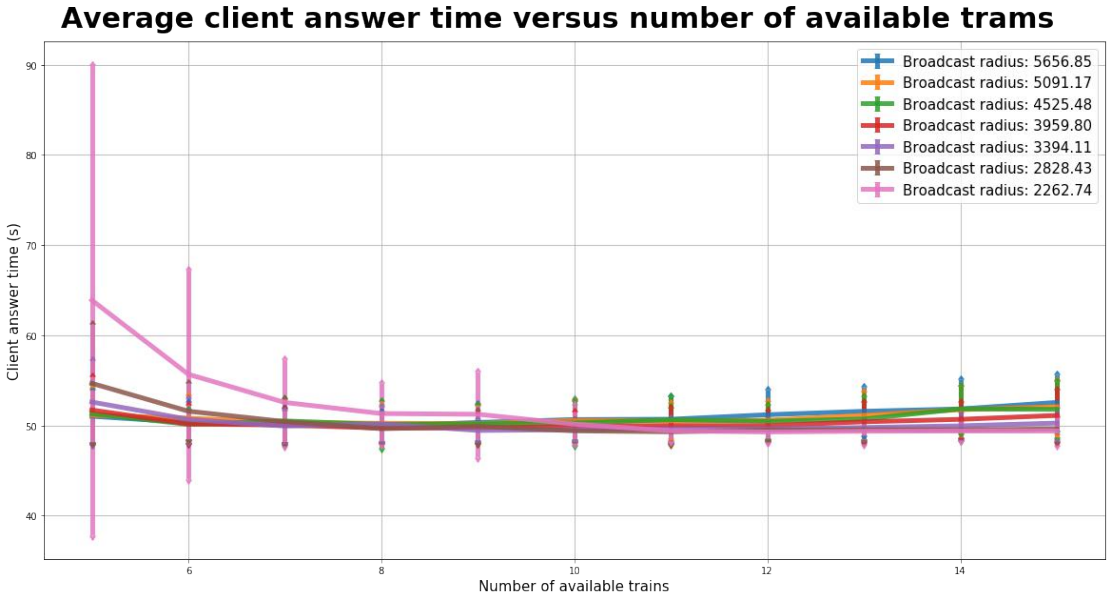


Figure 5.6: The image shows the average multiple clients mode answer time for each number of trams and broadcast range combination, portraying the standard deviation of the measurement in the error bars. The Y-Axis represents the answer time in seconds, and the X-axis the amount of available trams for the simulation.The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 60 experiments were carried out.

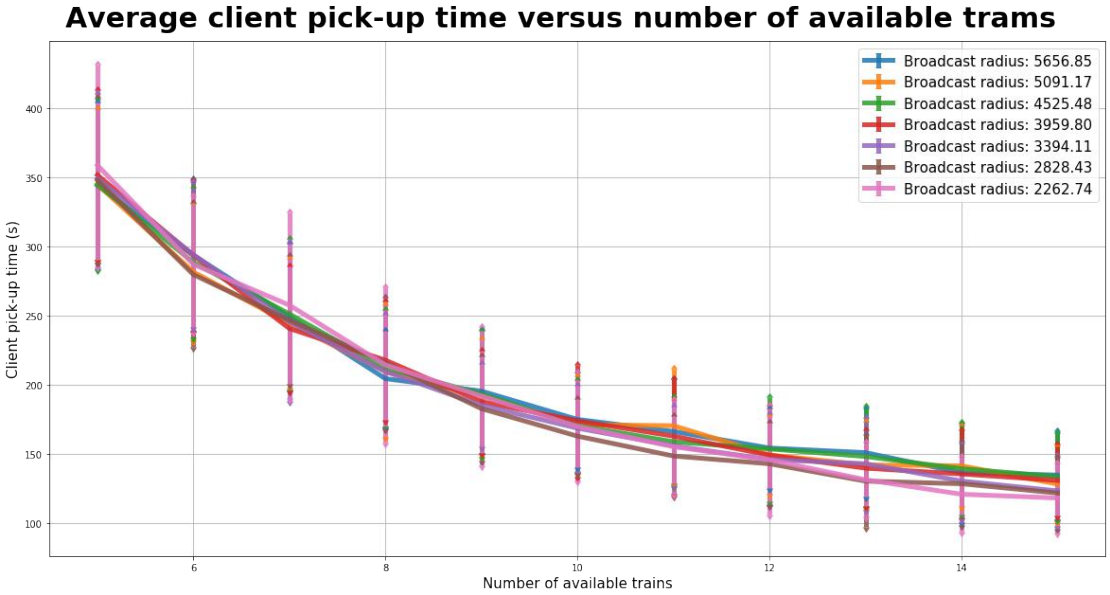


Figure 5.7: The image shows the average multiple clients mode pick-up time for each number of trams and broadcast range combination, portraying the standard deviation in the error bars. The Y-Axis represents the pick-up time in seconds, and the X-axis the amount of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 60 experiments were carried out.

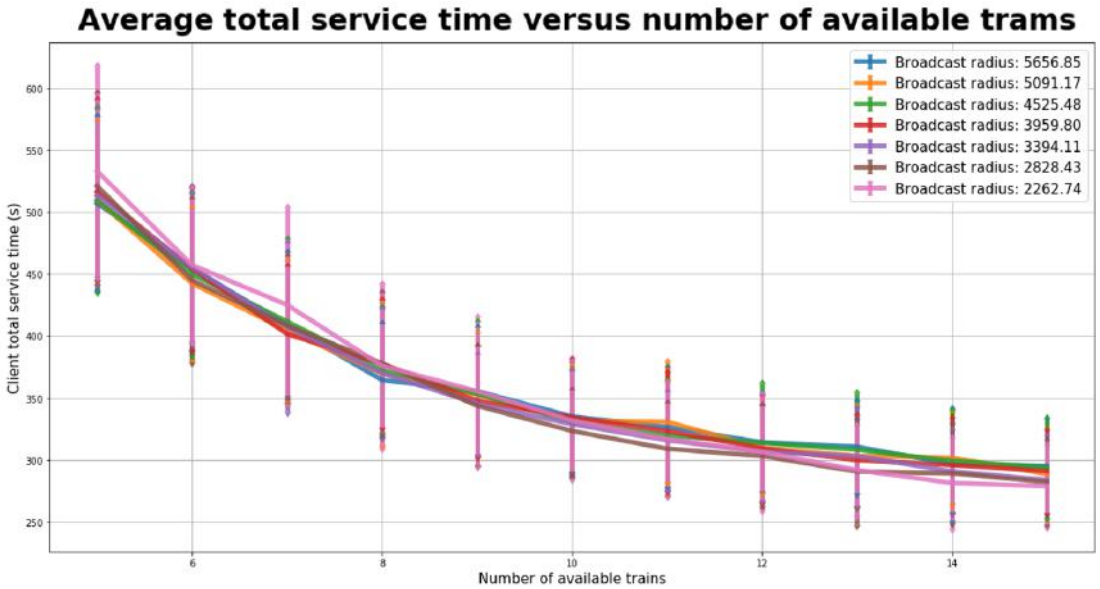


Figure 5.8: The image shows the average multiple clients mode total service time for each number of trams and broadcast range combination, portraying the standard deviation in the error bars. The Y-Axis represents the total service time in seconds, and the X-axis the amount of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 60 experiments were carried out.

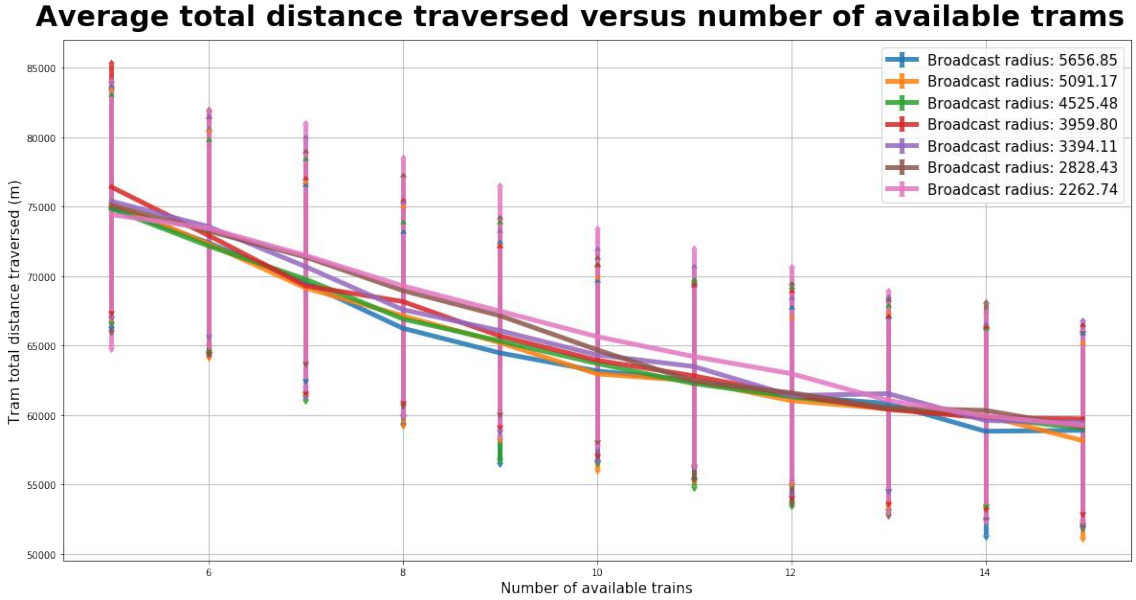


Figure 5.9: The image shows the average multiple clients mode total distance traversed for each number of trams and broadcast range combination, portraying the standard deviation in the error bars. The Y-Axis represents the distance in meters, and the X-axis the amount of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 60 experiments were carried out.

represents a clear increase in service quality. Secondly, that with regard to the system operation, barring a minimum value to guarantee that the system will work, the broadcast radius is not a very important factor. For all graphs, having the radius at 50% or 100% of the map diagonal does not overly affect the observed simulation parameters.

5.2 Simulation performance analysis

In order to discuss the system performance, one needs a formal measurement of it. An objective function is now proposed, to be used as metric to measure how the system performed under different circumstances.

The system wishes to maximize the service quality while minimizing the overall costs. Therefore the first step to defining the objective function is to create functions to measure the cost and quality.

Operating cost function

The system cost is influenced by several variables. In this work, the variables that will be considered are the tram broadcast radius fraction f_{dt}, the number of trams n_t and the total distance traversed d_t. The number of trams has an obvious effect on the cost. A larger broadcast radius implies that energy needs will be greater, increasing operational costs. Lastly, an increase in the total distance traversed for the same client requests corresponds to both a higher energy use and also higher maintenance costs in the long run. In order to normalize data from simulations with different number of clients requests, it was decided to use the total distance traversed per client $\frac{d_t}{n_c}$, where n_c is the number of clients serviced in the simulation.

The discussion in the previous paragraph can be summarized in the following cost function:

$$C = a_1 f_{dt} + a_2 n_t + a_3 \frac{d_t}{n_c} \tag{5.2}$$

where a_1, a_2 and a_3 are the weights of each parameter. The weights were chosen in

order to equilibrate the magnitude of the variables contemplated, bringing them to values around 10^2 and resulting in the function below.

$$C = 100f_{dt} + 20n_t + 0.1\frac{d_t}{n_c} \tag{5.3}$$

Service quality function

The service quality is given by the client waiting time. Thus this work the function proposed to measure quality is:

$$Q = t_t \tag{5.4}$$

and it is desired to minimize Q. The variable t_t is the average total service time. As seen above, Q depends on the variables $f_{dt}, n_t, \frac{d_t}{n_c}$, although the exact functional form of dependence is unknown.

The definition of two objective functions, both to be minimized by choice of the same decision variables, creates a multiobjective optimization problem. However, the goals are conflicting, since minimizing total service time tends to increase operational costs (and vice versa). This is handled in this monograph using the standard technique of minimizing a weighted sum of the two objective functions, with a positive weight λ which controls the trade-off between the two objectives. Thus, the overall objective function has the form:

$$\min\left(C + \lambda Q\right) \tag{5.5}$$

where λ is the trade-off parameter, which, depending on its value, gives more weight either to minimizing the cost function C or the quality function Q.

For the experiments, the code is run for a set of conditions, which are detailed in each experiment description. The optimizations here performed check, for each condition simulated, what is the objective function value, and determine the condition that has the lowest value. This is taken as the "optimal", for it has the better result for the particular objective function.

5.2.1 Optimization over number of trams and broadcast radius analysis

Applying the optimization function defined in equation 5.5 to the values observed in section 5.1 (5.1.1 and 5.1.2), the results shown in tables 5.3 and 5.4 were obtained, respectively for single and for multiple clients mode.

λ	**Number of trams**	**Tram radius fraction**	**Objective function**
0.01	15	1.5	276.6
0.05	15	1.2	309.6
0.1	15	1.2	350.5
0.3	15	1.2	514.0
0.5	13	1.2	676.1
0.8	13	1.2	913.6
1	13	1.2	1072
1.5	12	1.2	1464
2	9	1.2	1852
3	9	1.2	2612
4	9	1.2	3371
7	9	1.2	5648
10	9	1.2	7925
11	9	1.2	8684
12	9	1.2	9443
13	9	1.2	10202
14	9	1.2	10961
15	7	1.2	11715
16	7	1.2	12469
17	7	1.2	12469
18	7	1.2	12469
20	7	1.2	12469

Table 5.3: Table with the objective function values of equation 5.5 for single client mode. Simulations test the delivery of 15 clients, for a client arrival frequency parameter $f_c = 20$, having number of trams varying from 5 to 15 trams and tram broadcast radius fraction varying between 1.2 and 3.0 (actual radius from 6788.22 to 16970.55).

As expected from the graph results (in sections 5.1.1 and 5.1.2), one can observe that, at least for the proposed objective function, the broadcast radius does not influence the optimal value much, and the only case in which the client broadcast radius is different than 40% of the map diagonal is for $\lambda = 0.01$ in the single client mode.

λ	**Number of trams**	**Tram radius fraction**	**Objective function**
0.01	15	1.2	287.1
0.05	15	1.2	319.7
0.1	15	1.2	360.4
0.3	14	1.2	521.3
0.5	14	1.2	681.2
0.8	14	1.2	921.0
1	13	1.2	1079
1.5	10	1.2	1468
2	10	1.2	1847
3	8	1.2	2602
4	8	1.2	3344
7	5	1.2	5545
10	5	1.2	7693
11	5	1.2	8409
12	5	1.2	9126
13	5	1.2	9842
14	5	1.2	10558
15	5	1.2	11274
16	5	1.2	11990
17	5	1.2	12706
18	5	1.2	13422
20	5	1.2	14854

Table 5.4: Table with the objective function values of equation 5.5 for multiple clients mode. Simulations test the delivery of 15 clients, for frequency of clients $f_c = 20$, having number of trams varying from 5 to 15 trams and tram broadcast radius fraction varying between 1.2 and 3.0 (actual radius from 6788.22 to 16970.55).

From the tables it is clear that, for the situations considered, the single client mode has optimal value for $\lambda \leqslant 1.5$ and multiple clients mode is optimal for $\lambda \geqslant 2$ This leads to the non-intuitive conclusion that, for a higher emphasis on quality of service, it is better not to use ride sharing, although this gives the system the chance to deliver clients together, saving time. This claim is corroborated by the analysis of figure 5.4 in comparison with figure 5.8. The minimum total service time average value actually occurs in single client mode.

Before attempting to explain why single client mode is more efficient time wise, this claim needs to be evaluated further, especially because these simulations performed only have 15 clients each, which is not optimal for ride sharing between clients. This can also be deduced by the distance traversed graphs (figures 5.5 and

5.5): for the case of 15 clients there is hardly any difference between single and multiple client modes. Furthermore, the grid shaped map utilized in this experiment is also not the most suitable for ride sharing, since the probability that there is a client en route of the occupied tram is not as high with so many available stations and path possibilities.

The next experiment will increase the number of clients in the system, which should increase the probability of ride sharing between clients, providing better means of comparison between the two available modes.

5.2.2 Optimization over number of trams a broadcast radius analysis, for an operation of 60 clients

This section evaluates the system for a greater number of clients in each simulation. The analysis in the previous section was made for only 15 clients appearing in each simulation run. This might be enough to determine that the system functions, but not enough to evaluate the differences in performance between the single and multiple clients modes, especially considering the number of trams considered (that can go up to 15).

Therefore another test with more client appearances was carried out, this time for 60 clients per simulation instead of 15. The files in which these simulations are registered are *"Simulation Test - Number of trains by broadcast radius - Preset dataset - Grid - v3.4 - 60 clientes.ipynb"* for single client mode, and *"Simulation Test - Number of trains by broadcast radius - Preset dataset - Grid - v4.1 - 60 clientes.ipynb"*, for multiple clients mode. The parameters used are the same as shown in table 5.1. However, in this instance each simulation is repeated 20 times.

Figures 5.10 and 5.11 show the total service time measurements for single and multiple clients modes respectively. Figures 5.12 and 5.13 show the total distance traversed for each mode.

The first observation that can be made by the analysis of the total service time graphs is that, for a smaller number of trams, the multiple clients mode has the

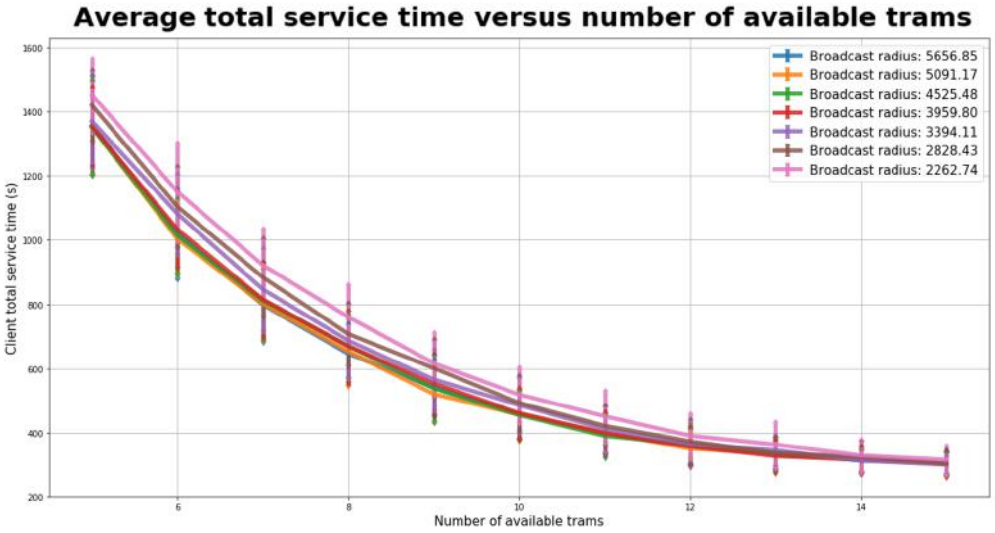


Figure 5.10: The image shows the average single client mode total service time for each number of trams and broadcast range combination, portraying the standard deviation in the error bars. The Y-Axis represents the total service time in seconds, and the X-axis the amount of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 20 experiments were carried out.

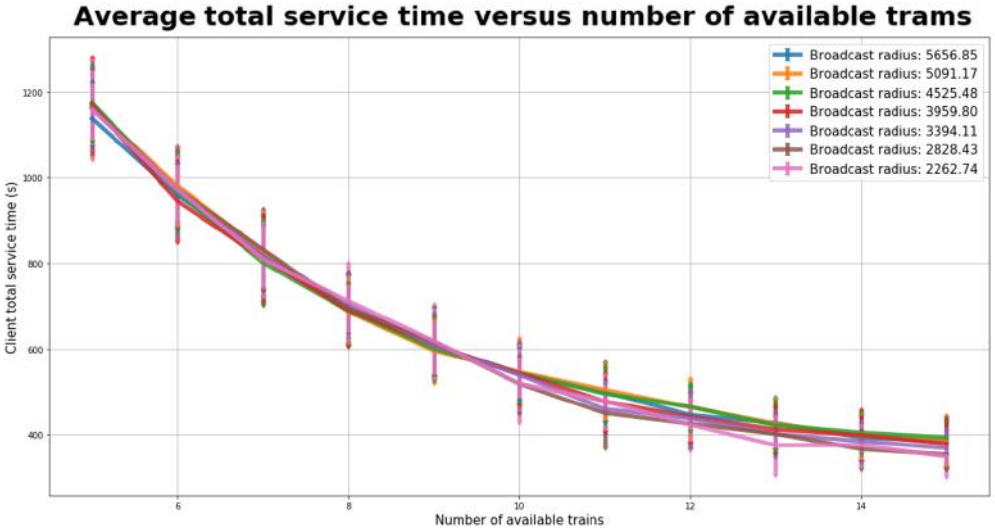


Figure 5.11: The image shows the average multiple clients mode total service time for each number of trams and broadcast range combination, portraying the standard deviation in the error bars. The Y-Axis represents the total service time in seconds, and the X-axis the amount of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 20 experiments were carried out.

higher quality (smaller time values). However for the higher number of trams the single client mode still shows itself as the most efficient time-wise.

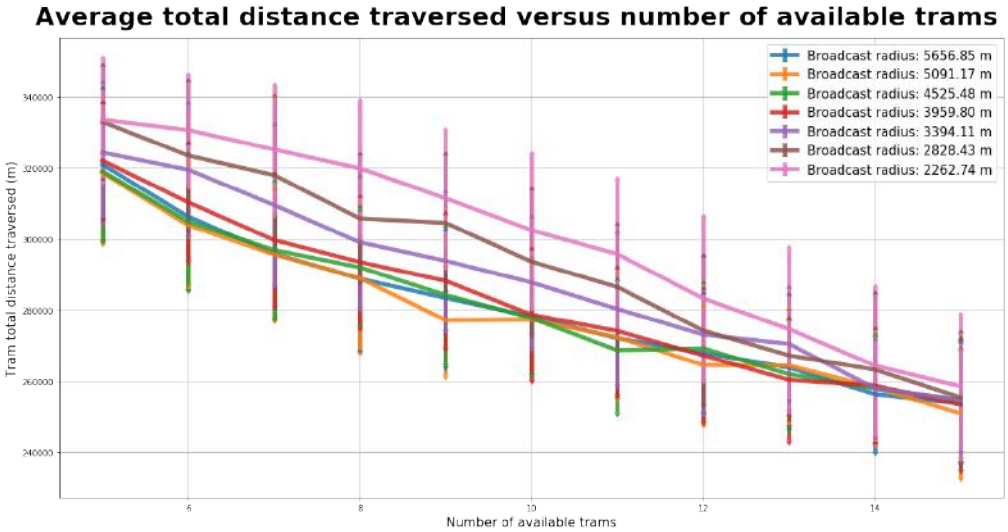


Figure 5.12: The image shows the average single client mode total distance traversed for each number of trams and broadcast range combination, portraying the standard deviation in the error bars. The Y-Axis represents the distance in meters, and the X-axis the amount of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 20 experiments were carried out.

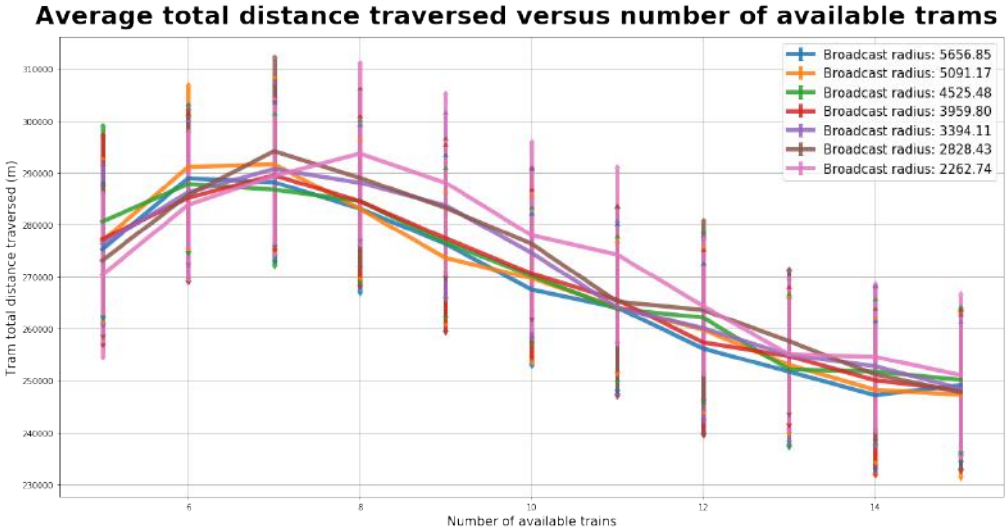


Figure 5.13: The image shows the average multiple clients mode total distance traversed for each number of trams and broadcast range combination, portraying the standard deviation in the error bars. The Y-Axis represents the distance in meters, and the X-axis the amount of available trams for the simulation. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 20 experiments were carried out.

The analysis of the total distance graphs shows, as expected, a clear overall decrease in the distance traversed when comparing the multiple clients mode to the

single client mode. It is noticeable, however, that this decrease is not as pronounced in the case of 15 trams per simulation. This leads to the suspicion that, for 15 trams, there is still not too much ride sharing going on. Also, for the multiple clients mode an anomalous behavior can be observed, for the increase of trams does not always corresponds to a decrease in the distance traversed. There is a maximum value around seven and eight trams.

The optimization results for these experiments can be seen in tables 5.5 and 5.6, for single and multiple clients modes respectively.

λ	**Number of trams**	**Tram radius fraction**	**Optimization value**
0.01	15	1.5	310.3
1.5	15	1.2	1573
2	12	1.2	1996
10	13	1.2	8663

Table 5.5: Table with the optimization results of equation 5.5 for single client mode. Simulations test the delivery of 60 clients, for frequency of clients $f_c = 20$, having number of trams varying from 5 to 15 trams and tram broadcast radius fraction varying between 1.2 and 3.0 (actual radius from 6788.22 to 16970.55).

λ	**Number of trams**	**Tram radius fraction**	**Optimization value**
0.01	15	1.2	356.8
1.5	13	1.2	1583
2	13	1.2	1985
10	5	1.2	7864

Table 5.6: Table with the optimization results of equation 5.5 for multiple clients mode. Simulations test the delivery of 60 clients, for frequency of clients $f_c = 20$, having number of trams varying from 5 to 15 trams and tram broadcast radius fraction varying between 1.2 and 3.0 (actual radius from 6788.22 to 16970.55).

These results corroborate the conclusions already seen in the previous section: for $\lambda \leqslant 1.5$ single client mode shows a better optimization, while for $\lambda \geqslant 2$ multiple clients mode is more efficient. That is, the single client mode is optimal when prioritizing service quality, while multiple clients mode is optimal when prioritizing cost reduction.

Also, once again, there is not much variation in the broadcast radius, even if in this experiment it can be observed a higher variation with respect to this parameter, especially in the distance traversed graphs.

However, in a real system the volume of the demand is not guaranteed to be constant, there can be times when there are more clients appearing and others in which there are very little in general. For this reason, this next experiment is designed to verify the system performance under different demand levels, which is modeled by the clients arrival frequency parameter f_c.

5.2.3 Optimization for different demand levels

This experiment was performed so that 60 clients appeared and were delivered in each simulation. It is registered in files *"Simulation Test - Number of trains by broadcast radius and client frequency - v3 - 60 clients.ipynb"*, for single client mode, and *"Simulation Test - Number of trains by broadcast radius and client frequency - v4 - 60 clients.ipynb"* for multiple clients mode. The parameters used in the experiment are shown in table 5.7. Each condition was repeated 20 times.

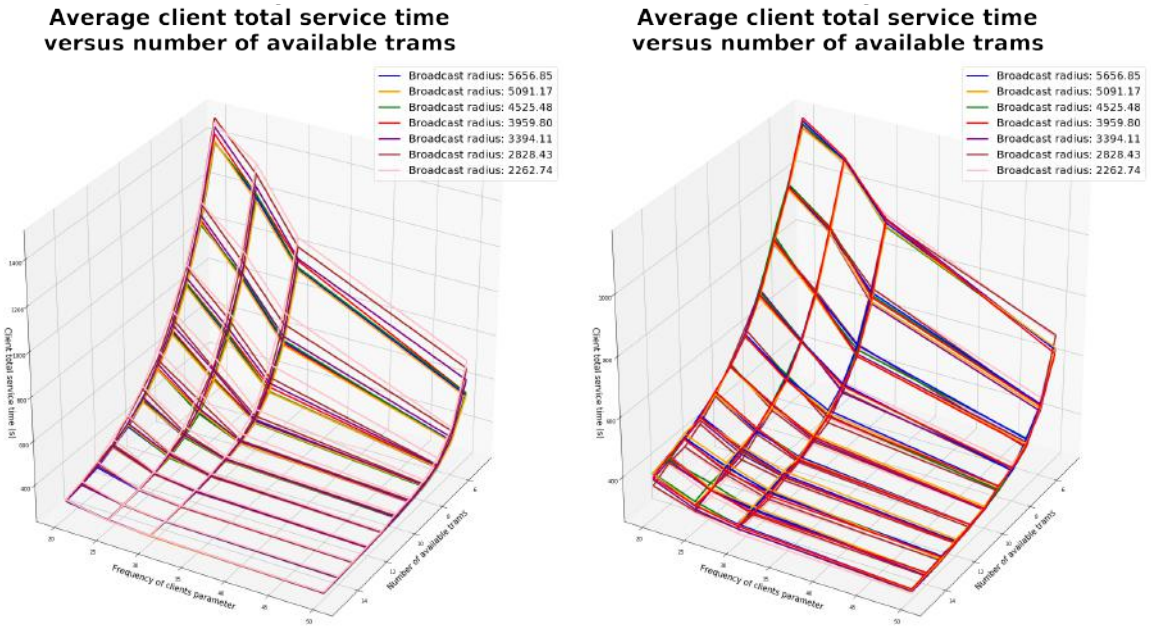
Client arrival dataset:	normal_map-grid_60rep_10000iter.np
Map:	GRID map (figure 4.8(c))
Tram speed:	20 m/s
Simulation step speed:	1 s/step
Election timeout:	45 s
Boarding time:	10 s
f_c **parameter:**	{20, 25, 30, 50}
Number of trams:	{5, 6, 7,8, 9, 10, 11, 12, 13, 14, 15}
Client broadcast radius:	$f_d d_{\text{map}}$, $f_d \in \{40, 50, 60, 70, 80, 90, 100\}$
Tram radius:	3 times client radius

Table 5.7: Parameters used in the experiments to evaluate the system performance under different demand levels. The first set of parameters are the conditions shared by all simulations run. The client arrival frequency parameter, number of trams and broadcast radius are varied through simulations and therefore are given as a set of values. For the GRID map, $d_{\text{map}} = 5656.85$.

The experiments showed that, for f_c values smaller than 20, the simulation was not guaranteed to work for the parameters considered. It shows that, for the higher amount of trams and higher demand the election timeout parameter (discussed in section 4.3) has to be adjusted. This happens because the high amount of messages exchanged delays the reading of important messages, and two or more trams may

win the election because it timed out before they are able to read the messages that would guarantee consensus. The election timeout will be further explored in section 5.2.5.

Figure 5.14 shows the total service time results for both system modes of operation, single client mode on the left, and multiple clients mode on the right. The difference with respect to the previous plots shown is that these add another axis to account for the f_c parameter.



(a) Single client mode

(b) Multiple clients mode

Figure 5.14: The images show the total service time for single client (to the left) and multiple clients (to the right) modes of operation. The Z-axis shows the total service time, the X-axis the number of trams in the simulation, and the Y-axis the clients arrival frequency parameter (f_c). The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 20 simulations were carried out.

From figure 5.14 one can notice that once again the broadcast radius does not have a strong impact on service time. Furthermore, as would be expected there is a clear decrease in the total service time with the increase of the number of trams, and also with the decrease in the client demand (less clients for a given time interval).

What is not so easy to observe, but is also of great importance is the comparison between single and multiple clients modes. To help in this analysis table 5.8 shows

the numerical values for a small sample of the conditions considered.

# of trams	Single client mode		Multiple clients mode	
	$f_c = 25$	$f_c = 50$	$f_c = 25$	$f_c = 50$
5	1341.5 ± 194.0	704.3 ± 151.9	1061.8 ± 122.3	675.4 ± 127.8
10	412.8 ± 66.7	300.0 ± 36.8	440.8 ± 64.3	309.8 ± 27.2
15	278.4 ± 12.9	261.4 ± 18.5	309.0 ± 28.6	267.0 ± 19.7

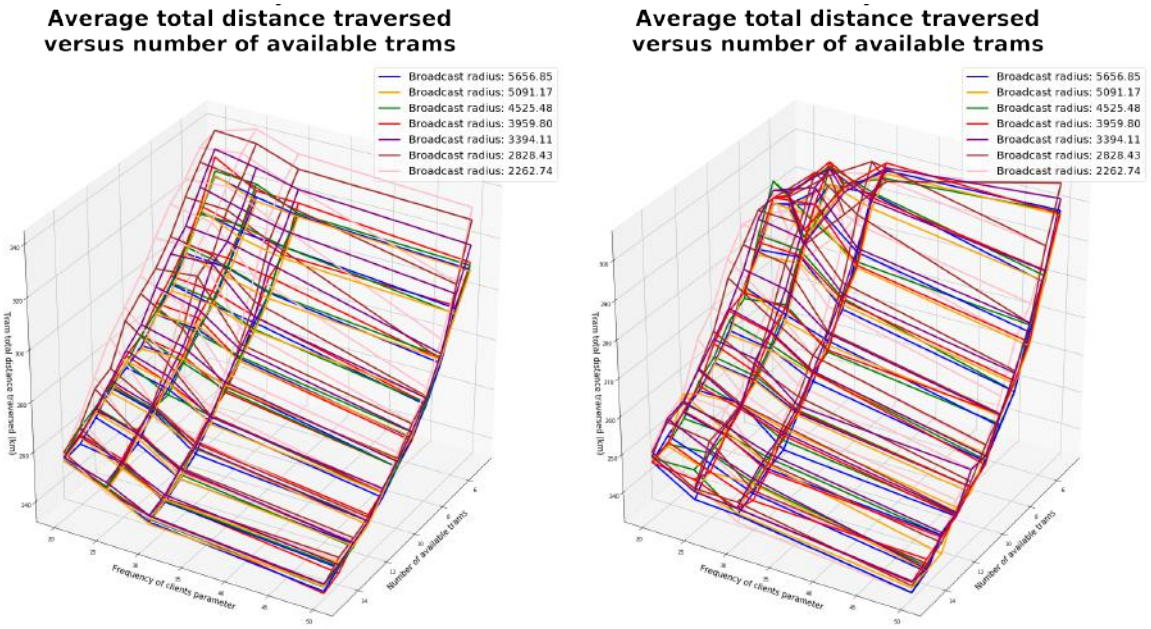
Table 5.8: Total service time values for client broadcast radius of 40% of the map diagonal (2262 m). The values show the average and the standard deviation of this parameter for the cases of 5, 10 and 15 number of trams available and client arrival frequency parameter of $f_c = 25$ and $f_c = 50$, for both modes of operation.

The results show, as with the previous experiments with $f_c = 20$, that the multiple clients mode is more efficient for a smaller of trams, while the single client mode shows better results for more trams, although in the case of $f_c = 50$ the difference in average is less than the uncertainty of the parameters. A possible reason is that the higher occupancy of the system by clients (which happens when there are less trams to service the clients) makes the multiple clients mode more efficient between the two.

Figure 5.15 shows the total distance traversed results for both modes of operation. The graphs are not as informative as the total service time ones, but it can be observed that, with relation to $f_c = 20$, the increase in f_c appears to decrease the differences in tram distances traversed between different client broadcast radius values, leading to the perception that the broadcast radius is less important with a smaller demand rate.

Also clear is that the multiple clients mode has a higher efficiency with respect to this particular parameter, as would be expected. There is, however, the notable exception of the case of 15 trams and $f_c = 50$. In this case there is hardly any difference between modes, and the increase in number of trams and f_c converges to that. This leads to the conclusion that in these conditions there is probably not much ride sharing occurring, which means that there is no difference between the modes.

It is also of note that, as the client demand decreases, the point of maximum



(a) Single client mode

(b) Multiple clients mode

Figure 5.15: The images show the total distance traversed for single client (to the left) and multiple clients (to the right) modes of operation. The Z-axis shows the total service time, the X-axis the number of trams in the simulation, and the Y-axis the clients arrival frequency parameter (f_c). The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 20 simulations were carried out.

observed for $f_c = 20$ becomes less and less pronounced, until the shape of the distance traversed surface becomes similar to that of the single clients mode surface.

Lastly, the optimization equation is applied over these values, obtaining the results shown in table 5.9, for single client mode, and table 5.10, for multiple clients mode.

The results show the same overall behavior observed for $f_c = 20$, in the sense that the broadcast radius does not suffer much variation with the change of the trade-off parameter λ and the single client mode is optimal when prioritizing service quality, while multiple clients mode is optimal when prioritizing cost reduction.

	$f_c = 25$			$f_c = 30$			$f_c = 50$		
λ	n_t	f_{dt}	O.F.	n_t	f_{dt}	O.F.	n_t	f_{dt}	O.F.
0.01	15	1.2	286.7	15	2.7	275.9	15	2.1	265.3
1.5	13	1.2	1513	13	1.2	1476	11	1.2	1421
2	13	1.2	1919	11	1.2	1871	11	1.2	1801
10	12	1.2	8366	10	1.2	8099	8	1.2	7754

Table 5.9: Table with the optimization results for single clients mode, with client arrival frequency parameter varying between 25, 30 and 50, based on equation 5.5. n_t is equivalent to the number of trams, which varies between 5 and 15, f_{dt} to the tram broadcast radius fraction, which varies between 1.2 and 3.0 (actual radius from 6788.22 to 16970.55), and *O.F.* to the minimum objective function value obtained.

	$f_c = 25$			$f_c = 30$			$f_c = 50$		
λ	n_t	f_{dt}	O.F.	n_t	f_{dt}	O.F.	n_t	f_{dt}	O.F.
0.01	15	1.2	317.3	15	1.2	292.6	15	1.2	275.1
1.5	12	1.2	1540	13	1.2	1480	10	1.2	1425
2	12	1.2	1935	13	1.2	1874	10	1.2	1796
10	5	1.2	7914	10	1.2	7971	7	1.2	7692

Table 5.10: Table with the optimization results for multiple clients mode, with client arrival frequency parameter varying between 25, 30 and 50, based on equation 5.5. n_t is equivalent to the number of trams, which varies between 5 and 15, f_{dt} to the tram broadcast radius fraction, which varies between 1.2 and 3.0 (actual radius from 6788.22 to 16970.55), and *O.F.* to the minimum objective function value obtained for a given trade-off parameter λ.

5.2.4 Experiment using the Dog map

This next experiment repeats the experiment in section 5.2.2, but for another map, to compare the results obtained. The map used this time will be the Dog map, shown in figure 4.8(b), and the dataset was changed to *normal_mapBig_60rep_-2000iter.npz* in order to accommodate the new map settings. The overall parameters used are shown in table 5.11.

As with the previous experiment, 60 clients appear and are serviced. Each simulation condition is repeated 20 times, in order to be able to compute statistics. The tram broadcast radius used was once again three times the client one, which is still enough margin to satisfy inequality 4.1, considering that the client broadcast radius varies between 1409 m and 3523 m in the selected map. The simulations are registered in the iPython notebooks *"Simulation Test - Number of trains by broadcast radius - Preset dataset - Dog - v3.4 - 60 clientes.ipynb"* and *"Simulation Test - Num-*

Client arrival dataset:	normal_mapBig_60rep_2000iter.npz
Map:	Dog map (figure 4.8(b))
f_c **parameter:**	20
Tram speed:	20 m/s
Simulation step speed:	1 s/step
Election timeout:	45 s
Boarding time:	10 s

Number of trams:	{5, 6, 7,8, 9, 10, 11, 12, 13, 14, 15}
Client broadcast radius:	$f_d d_{\text{map}}$, $f_d \in \{0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\}$
Tram radius:	3 times client radius

Table 5.11: Parameters used in the experiment to verify system performance in another map. The first set of parameters are the conditions shared by all simulations run. The number of trams and broadcast radius are varied through simulations and therefore are given as a set of values. For the Dog map, $d_{\text{map}} = 3522.78$.

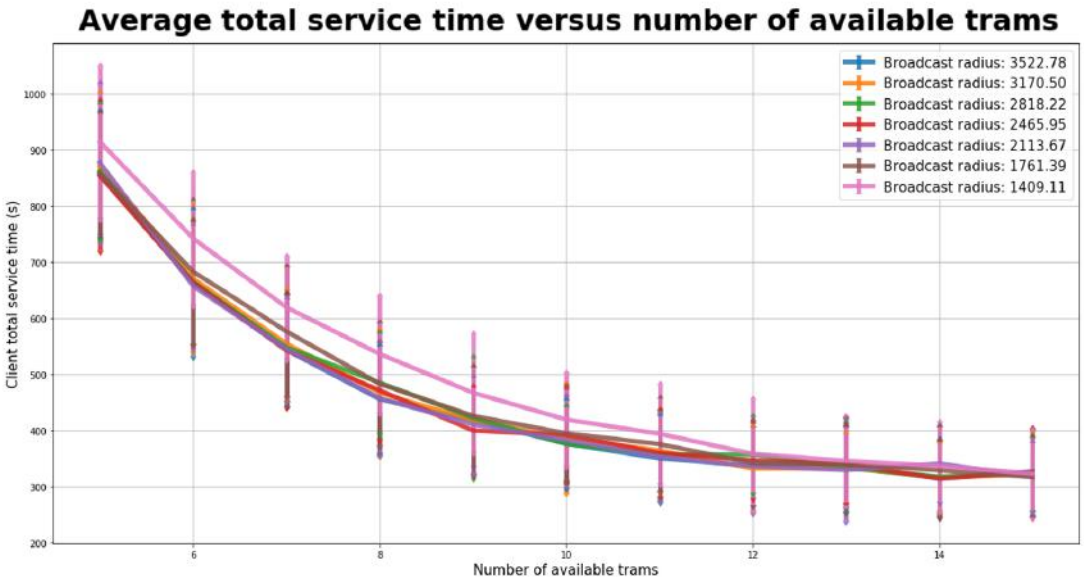
ber of trains by broadcast radius - Preset dataset - Dog - v4.1 - 60 clientes.ipynb".

Figure 5.16 shows the total service results for each mode of operation. Figure 5.17 shows the total distance traversed results for each mode.

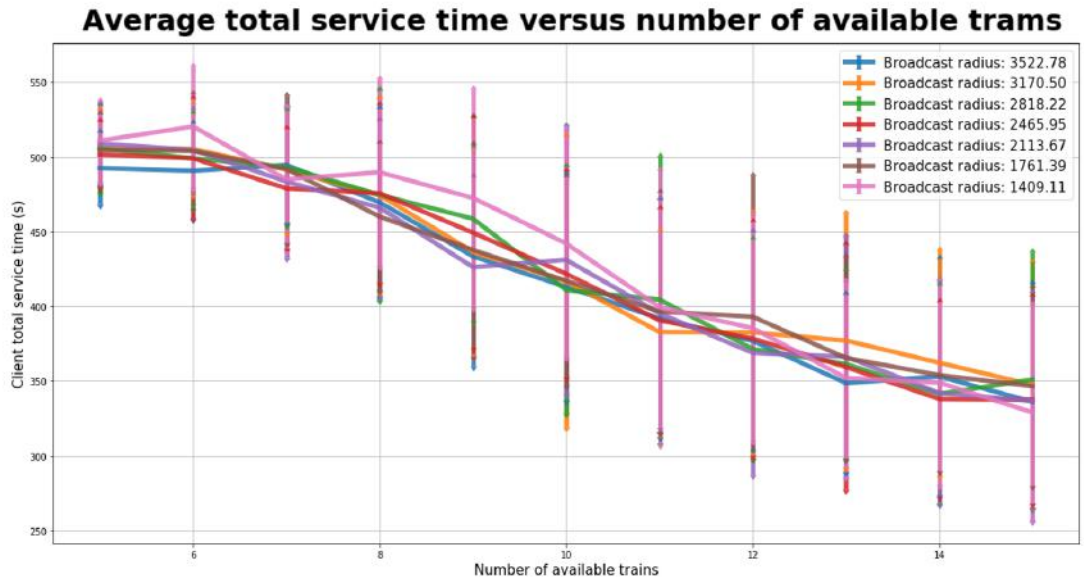
For the single client mode, the behavior of the total service time is in accordance with what was seen in figure 5.10: a monotonic decay with the increase in the number of trams. The time values are smaller in this map, but this is expected, since the map itself has smaller distances. The multiple clients mode deviates from the previously seen format (figure 5.11), oscillating more, as well as presenting higher uncertainties.

For the distance traversed, the multiple clients mode presents even more anomalous behavior, for the case with 15 trams no longer is the situation which has the smallest distance traversed. Considering the same figure for the previous map 5.13, and the pattern observed in the experiment that altered the frequency of clients, figure 5.15, one can imagine that this is a more pronounced version of the behavior observed when the demand is increased. Although the demand was the same as in figure 5.13, the map itself is much smaller, which increases the system occupancy.

It is supposed that the unexpected behavior is in fact related to the overall size of the map, and the nature of the multiple clients mode. The distances considered are smaller, and there are much less stations available. Consequently there are much



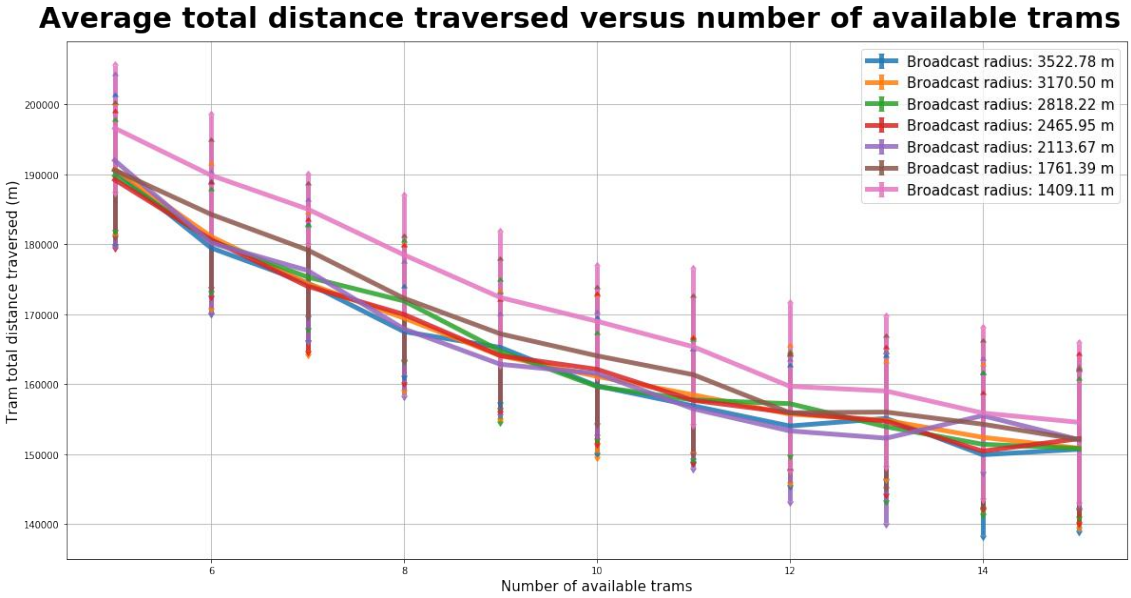
(a) Single client mode



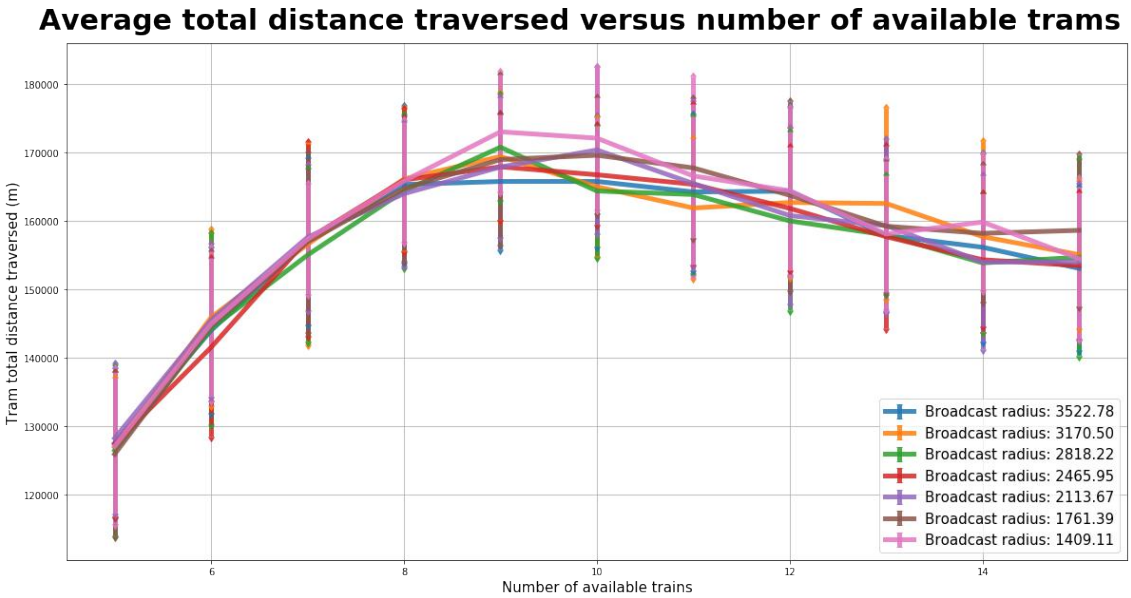
(b) Multiple clients mode

Figure 5.16: Total service time for single client (on top) and multiple clients (on the bottom) modes of operation, for the experiment suing the Dog map.These were made for $f_c = 20$ and 60 clients delivered. The Y-axis represents the time in seconds, and the X-axis the number of trams considered. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 20 simulations were carried out.

less client route possibilities. Therefore, with a lower amount of trams the system is able to optimize ride sharing, for there is a high probability the the clients are already on the path that trams are going to pass through. With the increase in the number of trams there is a decrease in the trams occupancy, which corresponds to a



(a) Single client mode



(b) Multiple clients mode

Figure 5.17: Total distance traversed for single client (on top) and multiple clients (on the bottom) modes of operation, for the experiment suing the Dog map.These were made for $f_c = 20$ and 60 clients delivered. The Y-axis represents the distance in meters, and the X-axis the number of trams considered. The graphs are color-coded according to client broadcast radius, with the color code appearing in the inset (top right). For each setting of broadcast radius and number of trams, 20 simulations were carried out.

higher probability that sharing the ride with some other client will not correspond to the faster way to service the client. With the decrease of ride sharing, there is an increase in the distance traversed, accounting for the curves observed in figure

5.17(b).

The value of the objective function was calculated for the conditions tested, with the smallest values for each condition being shown in table 5.12.

	Single Client			**Multiple Clients**		
λ	n_t	f_{dt}	Obj.	n_t	f_{dt}	Obj.
0.01	14	2.1	322.1	15	1.2	335.8
1.5	12	1.2	1298	5	1.2	1158
2	12	1.2	1611	5	1.2	1374
10	7	1.2	6304	5	1.2	4828

Table 5.12: Optimization results for Dog map experiment, considering both modes of operation. The frequency of clients parameter was set at $f_c = 20$, number of trams, shown as n_t, was varied between 5 and 15, tram broadcast radius fraction, shown as f_{dt}, was varied between 1.2 and 3.0 (actual radius from 4227.33 to 10568.34). *Obj.* is the minimum optimization value obtained for a given λ, which is the trade-off parameter. 60 clients are serviced for each simulation, and each condition was repeated 20 times.

The results lead to the conclusion that this map shows itself to be more favorable for multiple clients mode. The pattern of single client mode having higher quality maintains itself, but the difference between modes is less pronounced in this case, with multiple clients mode surpassing single clients mode sooner than in previous experiments. In this case, single clients mode is only optimal for maximum service quality ($\lambda = 0.01$), that is, minimum Q value.

It is noticeable that the map choice has a high impact on the overall results obtained.

5.2.5 Election timeout parameter analysis

This experiment is designed to ascertain the minimum election timeout value the system can have and still function.

The experiment will test the system for five different client arrival frequency parameters: 25, 20, 15, 10 and 5. The verification of a certain timeout value is considered successful if the experiment runs 50 simulations and no error occurs, with each simulation delivering 60 clients.

The sets of parameters used in this experiment are shown in table 5.13. The

broadcast radius was chosen as 40% of the map diagonal because almost all of the optimizations performed chose this parameter as the optimal. The number of trams was chosen as 15 since a greater amount of trams implies a larger number of message exchanges, putting the election timeout under stress.

Client arrival dataset:	normal_map-grid_60rep_2000iter.npz
Map:	GRID map (figure 4.8(c))
Number of trams:	15
Client broadcast radius:	$0.4d_{\text{map}} = 2262.74\,\text{m}$
Tram radius:	3 times client radius = 6794.22 m
Tram speed:	20 m/s
Simulation step speed:	1 s/step
Boarding time:	10 s
f_c **parameter:**	{25, 20, 15, 10, 5}
Initial election timeout:	{45, 45, 50, 50, 70}, respectively

Table 5.13: Parameters used in the experiment to evaluate election timeout parameter behavior. The first set of parameters are the conditions shared by all simulations run. The client arrival frequency parameter and the initial timeout values are varied through simulations and therefore are given as sets of values.

The timeout is started at a specified value, shown in the table and it diminished by five seconds at each successful verification, until an error occurs. When it does, it is determined that this is the limit of the election timeout for such demand, and the minimum value is taken to be equal to the last feasible value of the timeout.

The experiment was performed on iPython notebooks *"Simulation - Election timeout parameter - v3.ipynb"* and *"Simulation - Election timeout parameter - v4.ipynb"*. The results can be observed in table 5.14.

	Election timeout (s)	
f_c	**Single Client**	**Multiple Clients**
25	15	25
20	20	25
15	20	25
10	20	25
5	65	50

Table 5.14: Minimum election timeout values for 15 trams and client broadcast radius of 40% of the map diagonal, with uncertainty of 5 s. Each test repeats the simulation 50 times.

As expected, the increase in demand makes it necessary to have a bigger election timeout. It is also observed that in general the single client mode required a smaller timeout value in order to function correctly.

In general, the election timeout values are lower than the 45 s used in other experiments, however the increase in the broadcast radius increases also the rate of message exchanges, which increases the needed election timeout.

Chapter 6

Concluding remarks

This work has shown how to carry out the theoretical and practical automation of a dedicated transportation system with distributed coordination, in the form of automata models and a simulation engine. The engine implements a shared autonomous vehicle system (SAV) in two modes of operation: single client mode, in which each tram services only one client at a time; and multiple clients mode, in which there is the possibility of ride sharing between clients. The simulation was tested and its performance analyzed, verifying that it can be used to optimize operational costs and quality of service through the choice of system parameters and detect some failure cases to compute the feasibility limits of said system parameters. Results consistently showed that, for the proposed objective function, while the multiple clients mode is the better option to reduce system costs, the single client mode is the better option to increase service quality.

The multiple clients mode can show worse service quality results, considering higher quality as less time waited by the clients, which is not an intuitive result, for common sense says that sharing rides should give the trams the opportunity to service clients faster. The results show that this is the case when client occupancy with respect to the number of trams is higher. When there are less trams to service clients, multiple clients mode is optimal in both quality improvement and cost reduction. However for more trams, and therefore a lower tram per client ratio, the system is not as stressed, and single client mode means that there are no stops

occurring during client servicing to pick up and/or drop off clients, which in turn reduces the average total service time.

However, there are still further analyses that can be made, such as the level of usage of the ride sharing functionality and level of idleness of trams. In several experiments, underutilization of the multiple clients mode was conjectured, but this needs to be confirmed by the analysis of the actual number of shared rides in a simulation. Furthermore, it is important to take into account how often trams are stagnant in a simulation, and especially compare the tram idleness ratio with respect to the number of trams available.

It would also be interesting to test system behavior with different client demand models, since only the uniform distribution was used and it does not usually correspond to what is observed in practical systems, and different choices of objective function, with further analysis of the sensitivity observed in the functions employed with respect to the system parameters changed. This monograph used ad hoc optimization techniques; however black box optimization methods should be considered for further studies, such as the Nelder-Mead algorithm [16].

Furthermore, there are some real world considerations that have not been taken into account in the model implemented. For example, the system does not consider the possibility of client withdrawal, that is, the cancellation of a client request before it has been serviced. Also, no road blockages or traffic conditions have been taken into account, nor has it been considered that trams might malfunction during operation.

Lastly, it has been observed that the proposed system has a scalability problem: as the map distances increase (and consequently the map diagonal, used as reference in the system), the broadcast radius must also increase, which is impractical. A possible solution is to divide the map into zones, with trams assigned to specific zones. Each zone could be operated as simulated in this project, and a system similar to the one proposed in [17], for example, could manage the tram distributions through zones.

Bibliography

[1] BERRADA, J., LEURENT, F. "Modeling Transportation Systems involving Autonomous Vehicles: A State of the Art", *Transportation Research Procedia*, v. 27, pp. 215–221, 01 2017. doi: 10.1016/j.trpro.2017.12.077.

[2] LIU, J., JONES, S., ADANU, E. K. "Challenging human driver taxis with shared autonomous vehicles: a case study of Chicago", *Transportation Letters*, v. 0, n. 0, pp. 1–5, 2019. doi: 10.1080/19427867.2019.1694202.

[3] VAN STEEN, M., TANENBAUM, A. S. *Distributed Systems*. distributed-systems.net, 2017.

[4] GARCIA-MOLINA, H. "Elections in a Distributed Computing System", *IEEE Transactions on Computers*, v. c-31, n. 1, pp. 48–59, 1982.

[5] CANCA, D., BARRENA, E., ALGABA, E., et al. "Design and analysis of demand-adapted railway timetables", *Journal of Advanced Transportation*, v. 48, pp. 119137, 2014. doi: 10.1002/atr.1261.

[6] "Uber Newsroom, Optimizing a dispatch system using an AI simulation framework". `https://www.uber.com/newsroom/semi-automated-science-using-an-ai-simulation-framework`, 2014. Accessed in: 2019-11-01.

[7] DE OLIVEIRA, A. C. N. "VA Project Repository". `https://github.com/AmieOliveira/V.A..git`, 2019. Accessed in: 2019-12-12.

[8] DIJKSTRA, E. W. "A Note on Two Problems in Connection with Graphs", *Numerische Mathematik*, v. 1, pp. 269–271, 1959. doi: 10.1007/BF01386390.

[9] VAN STEEN, M., TANENBAUM, A. S. "Election algorithms". In: *Distributed Systems*, 3rd ed., cap. 6, distributed-systems.net, 2017.

[10] CASSANDRAS, C. G., LAFORTUNE, S. "Petri Nets". In: *Introduction to Discrete Event Systems*, 2 ed., cap. 4, New York, Springer, 2008.

[11] HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 2001.

[12] CASSANDRAS, C. G., LAFORTUNE, S. "Languages and Automata". In: *Introduction to Discrete Event Systems*, 2 ed., cap. 2, New York, Springer, 2008.

[13] "NetworkX – Software for complex networks". `https://networkx.github.io`, 2019. Accessed in: 2019-12-14.

[14] "random – Generate pseudo-random numbers". `https://docs.python.org/3.7/library/random.html`, 2019. Accessed in: 2019-12-26.

[15] "matplotlib". `https://matplotlib.org/3.1.0/index.html`, 2019. Accessed in: 2019-12-26.

[16] GAVIN, H. P. "The Nelder-Mead Algorithm in Two Dimensions", Institution: Duke University. Available at `http://people.duke.edu/~hpgavin/cee201/Nelder-Mead-2D.pdf`. Accessed in: 2020-01-08, 2016.

[17] MIAO, F., LIN, S., MUNIR, S., et al. "Taxi Dispatch with Real-Time Sensing Data in Metropolitan Areas a Receding Horizon Control Approach *", *IEEE Transactions on Automation Science and Engineering*, v. 13, 04 2015. doi: 10.1145/2735960.2735961.

Appendix A

Dijkstra algorithm pseudo-code

Algorithm 1 Dijkstra's Shortest Path First Algorithm

```
1:  function DIJKSTRA(graph, source, target)  ▷ Algorithm to find shortest path from source to target in graph. Returns distance and path.
2:      Q = vertexSet()  ▷ This vertex set stores all the graph vertexes that have not yet been analyzed by the algorithm
3:
4:      for each vertex v in graph do
5:          dist[v] = INFINITY
6:          prev[v] = UNDEFINED
7:          Q.add(v)
8:      end for
9:      dist[v] = 0
10:
11:     while Q not EMPTY do
12:         u = Q.minDistVertex()  ▷ Get the vertex v with minimum dist[v] from the ones in Q
13:
14:         Q.remove(u)
15:
16:         if u == target then
17:             break
18:         end if
19:
20:         for each vertex v in neighbors(u) do ▷ Only vertices v that are still in Q
21:             alt = dist[u] + length(u, v)
22:             if alt < dist[v] then
23:                 dist[v] = alt
24:                 prev[v] = u
25:             end if
26:         end for
27:     end while
```

```
28:                                                  ▷ Recover path to target
29:     s = sequence()                                          ▷ Empty sequence
30:     u = target
31:     if (prev[u] not UNDEFINED) or (u == source) then        ▷ Only do
    something if vertex is reachable
32:         while u not UNDEFINED do
33:             s.push(u)                ▷ Put vertex in the beginning of the sequence
34:             u = prev[u]
35:         end while
36:     end if
37:
        return dist[target], s
38: end function
```

Appendix B

Election algorithm pseudo-code

Algorithm 2 Election Algorithm

```
1: function ELECTION(t, c)  ▷ Tram t executes this election procedure when it receives a service request by client c
2:     client = c.ID
3:     distance = calculateRoute(t.position, c.position)  ▷ Function "calculateRoute" will answer the total distance from the tram t current position to the client c pick-up location, passing through t's route plan.
4:     send("request received", c)  ▷ t lets c know it has received request and is proceding to process it
5:                                  ▷ Setting timer to start elections
6:     stop_time = t.time + t.start_timeout  ▷ t.time is the tram clock time, and start_timeout is a random time specific to tram t that specifies how long t will wait before sending election messages
7:     while t.time < stop_time do
8:         if ( message_received.type == "leader" ) and ( message_received.client == c ) then
9:             return  ▷ Lost Elections
10:        end if
11:        if ( message_received.type == "election" ) and ( message_received.client == c ) then
12:            if ( message_received.distance < distance ) then
13:                return  ▷ Lost Elections
14:            end if
15:            if ( message_received.distance == distance ) and ( message_received.sender.ID > t.ID ) then
16:                return  ▷ Lost Elections
17:            end if
18:            send("won round", message_received.sender, client=client)  ▷ Tram t sends message back to let sender know it has lost against t and should drop out of elections
```

```
19:             break
20:         end if
21:     end while
22:     send("election", distance=distance, client=client)
23:
24:                                                 ▷ Setting timer to finish elections
25:     stop_time = t.time + t.finish_timeout                    ▷ finish_timeout
    is a predefined variable that stipulates for how long a tram should wait for an
    answer to the election message sent
26:     while t.time < stop_time do
27:         if ( message_received.type == "leader" ) and ( message_received.client
    == c ) then
28:             return                                                ▷ Lost Elections
29:         end if
30:         if ( message_received.type == "election" ) and ( message_re-
    ceived.client == c ) then
31:             if ( message_received.distance < distance ) then
32:                 return                                            ▷ Lost Elections
33:             end if
34:             if ( message_received.distance == distance ) and ( message_-
    received.sender.ID > t.ID ) then
35:                 return                                            ▷ Lost Elections
36:             end if
37:         end if
38:         send("won round", message_received.sender, client=client)          ▷
    Tram t sends message back to let sender know it has lost against t and should
    drop out of elections
39:     end while
40:
41:                 ▷ If tram t had the timeout (left the while loop) and did not lose
    the elections, it has won them. Therefore t will proceed to let other trams and
    client know the result
42:     send("leader", client=client)
43:     send("request accepted", c)
44:     t.clientList.add(c)                                                ▷
45:     return
46: end function
```