UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de simulaciones de transporte adaptado a la demanda con técnicas de Inteligencia Artificial

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Pérez Sabater, Marc

Tutor/a: Julian Inglada, Vicente Javier

Director/a Experimental: JORDAN PRUNERA, JAUME MAGI

CURSO ACADÉMICO: 2021/2022

Resum

El transport a les ciutats és un element del tot quotidià, el qual utilitza o ha utilitzat qualsevol persona. Des de vehicles particulars, fins taxis o autobusos, passant pels nous vehicles de lloguer i serveis de xòfers a domicili. Tot i això, el fet que utilitzem el transport quasi tots els dies no implica que el seu ús estiga optimitzat o almenys ben aprofitat.

En aquest treball es documenta la modificació del simulador SimFleet per a poder utilitzar un mecanisme de mobilitat urbana basat en la demanda, de forma que la ruta del transport es decideix de forma dinàmica en funció de com és la demanda de viatges. SimFleet és un simulador de flotes basat en agents per a testejar estratègies. Es descriuen les modificacions realitzades a un software existent per aconseguir el resultat dels experiments realitzats. L'objectiu és comprovar la millora o empitjorament de les prestacions d'un vehicle de transport sota demanda, al ampliar més o menys la seua capacitat, i utilitzant en cada cas distintes estratègies.

Paraules clau: Sistema multi-agent, transport urbà, vehicles intel·ligents, simulador, flota urbana, mobilitat compartida, resposta a la demanda, simulació

Resumen

El transporte en las ciudades es un elemento totalmente cotidiano, el cual utiliza o ha utilizado cualquier persona. Desde vehículos particulares, hasta taxis o autobuses, pasando por los nuevos vehículos de alquiler y servicios de chóferes a domicilio. Sin embargo, el hecho de que utilicemos el transporte casi todos los días no implica que su uso esté optimizado o al menos bien aprovechado.

En este trabajo se documenta la modificación del simulador SimFleet para poder utilizar un mecanismo de movilidad urbana basado en la demanda, de modo que la ruta del transporte se decide de forma dinámica en función de cómo es la demanda de viajes. SimFleet es un simulador de flotas basado en agentes para testear estrategias. El objetivo es comprobar la mejora o empeoramiento de las prestaciones de un vehículo de transporte bajo demanda, al ampliar más o menos su capacidad, utilizando en cada caso distintas estrategias.

Palabras clave: Sistema multi-agente, transporte urbano, vehículos inteligentes, simulador, flota urbana, mobilidad compartida, respuesta a la demanda, simulación

Abstract

Transportation in cities is an everyday element that anyone uses or has used. From private vehicles to taxis or buses, to new rental vehicles and driver services at home. However, the fact that we use transport almost every day does not mean that its use is optimized or at least well exploited.

This work documents the modification of the SimFleet simulator to be able to use a demand-based urban mobility mechanism, so that the transport route is

decided dynamically based on how the travel demand is. SimFleet is an agent-based fleet simulator for testing strategies. The aim is to check the improvement or worsening of the performance of a transport vehicle on demand, by expanding more or less its capacity, and using in each case different strategies.

Key words: Multi-agent system, urban transport, smart vehicles, simulator, urban fleet, shared mobility, demand-responsive, simulation

Índex

Índex de figures

Índex de taules

CAPÍTOL 1
Introducció

El transport públic és hui en un dia una de les coses més comunes a les grans ciutats. Però com tot, sempre és millorable. En aquest document s'introduïx una sèrie d'experiments realitzats amb el simulador SimFleet, on es provaran distintes estratègies amb un vehicle adaptat a la demanda, per tal de comprovar quina és la més eficient o efectiva. A continuació, es desenvoluparan la motivació per a l'elecció del tema com a projecte, els objectius d'aquest i la seua estructura.

1.1 Motivació

La motivació per a realitzar aquest treball sorgeix de la necessitat de nous sistemes de mobilitat a les ciutats de tipus "demand-responsive", és a dir, de servei sota demanda.

La intenció és desenvolupar una solució per estalviar combustible del transport a les ciutats per fer front al gran problema de la contaminació urbana, de caràcter sensible i adaptatiu a la demanda. Es pretén crear un sistema que simule aquest tipus de vehicle per a poder avaluar-lo mitjançant diverses mètriques i establir les estratègies més eficients.

Una raó més per a realitzar aquest treball és la possibilitat, a llarg termini, de plasmar aquest tipus de vehicles o comportament intel·ligent en la realitat, en cas de demostrar una millora respecte a les prestacions actuals de serveis similars.

1.2 Objectius

L'objectiu principal d'aquest treball és desenvolupar un nou mecanisme de mobilitat sota demanda dins del simulador SimFleet i recollir i avaluar una serie de resultats dels experiments amb distintes estratègies a les simulacions. Per aconseguir això, es necessita realitzar els següents objectius:

- Revisar i estudiar l'estat de l'art del camp dels simuladors de tràfic urbà, per poder avaluar el punt actual i saber des d'on es parteix.

- Plantejar el desenvolupament o les modificacions del simulador que es trie per al treball, per a poder representar un sistema de mobilitat basat en la demanda.
- Desenvolupar el codi per a modificar el funcionament del simulador per a obtindre un sistema demand-responsive, i desenvolupar distintes estratègies per a la gestió de la demanda.
- Validar amb una serie de proves el sistema desenvolupat, amb suficients variants i valors realistes dels paràmetres, per a obtindre uns resultats significatius i amb cert sentit.
- Representar els resultats per a que es puguen entendre visualment i es mostre quina o quines són les millors estratègies per a cada combinació de paràmetres.
- Extraure conclusions en base als resultats obtinguts als experiments.

1.3 Estructura de la memòria

Aquest treball es dividirà en diverses parts. A grans trets, es comença parlant de l'estat de l'art (Capítol 2), on s'explica en profunditat el que és un sistema multi-agent i es presenta un context tecnològic amb diversos simuladors per a transport urbà i altres treballs acadèmics sobre temes similars. A continuació es mostra la solució proposada (Capítol 3), seguida del disseny d'aquesta. Dins del disseny (Capítol 4) es presenta SimFleet, la tecnologia utilitzada. Seguidament es pot veure el desenvolupament (Capítol 5) de la solució, amb les modificacions que s'han hagut de fer al codi existent de SimFleet i les estratègies proposades. Després es mostra la fase d'experimentació (Capítol 6), amb els seus resultats i una discussió per a raonar-los. Finalment es presenten les conclusions (Capítol 7) per a contrastar els objectius del principi, la relació amb els estudis cursats (Capítol 8), una proposta de treballs futurs (Capítol 9). Per últim, es presenta un xicotet glossari en un annex amb alguns termes tècnics utilitzats al llarg d'aquest document (Capítol A), el qual es recomana repassar ambs de llegir el treball.

CAPÍTOL 2
Estat de l'art

En aquest capítol es presenta l'estat de l'art de l'àmbit dels simuladors de transport amb sistemes multi-agent.

Així doncs, el primer que es descriu són els sistemes multi-agent, els distints enfocaments que hi ha sobre aquests i les seues característiques i avantatges. També es mencionen algunes ferramentes per al desenvolupament d'agents.

A continuació es presenten una serie de simuladors de transports, els quals s'han estudiat i determinat quin s'ajusta més a les necessitats del projecte.

Finalment també es descriuen alguns treballs acadèmics de temàtica similar a aquest.

2.1 Sistemes multi-agent

El domini dels sistemes multi-agent (SMA) és una ciència i una tècnica que tracta amb els sistemes d'intel·ligència artificial a la xarxa [2].

Els agents són generalment vistos com a entitats intel·ligents que existeixen dins d'un cert context o ambient, i que es poden comunicar mitjançant un mecanisme de comunicació entre processos, usualment un sistema de xarxa, utilitzant protocols de comunicació.

Un sistema multi-agent està constituït per un conjunt d'agents (dos o més agents) que interactuen els uns amb els altres. Per interactuar amb èxit, requereixen capacitats per cooperar, coordinar-se i negociar amb cadascun dels altres, tal com fa la gent.

Podem veure l'aplicació dels SMA al món real a aplicacions gràfiques com a jocs d'ordinador, en pel·lícules, per a sistemes de defensa coordinats, per al transport, la logística, els gràfics, sistemes d'informació geogràfica, diagnòstic, etc.

2.1.1. Funcionament

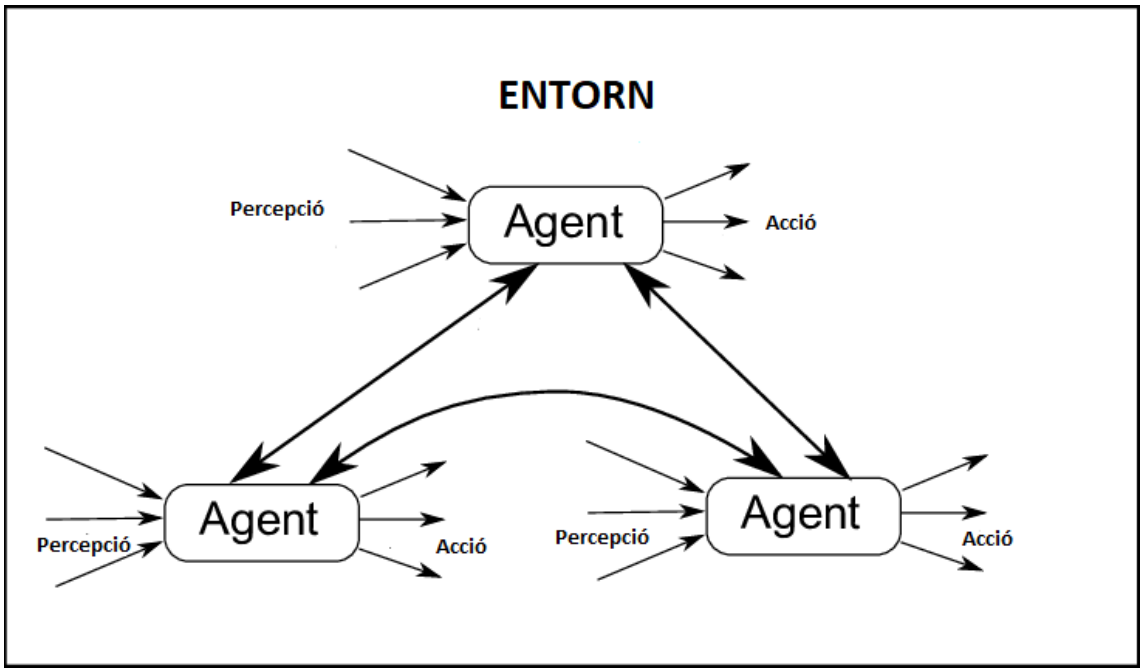
Cada agent busca satisfer els seus objectius, per a la qual cosa pren decisions, descompon els objectius en sub-objectius i executa feines. Per a acomplir aquestos

objectius necessita col·laboració amb altres agents, mitjançant negociació, delegació i coordinació [3].

Per a la realització d'aquesta comunicació amb altres agents s'utilitzen dos tipus de servicis: les pàgines blanques, per a buscar altres agents, i les pàgines grogues, un directori per cercar o registrar els serveis disponibles.

Les accions que realitza cada agent venen marcades per les percepcions que rep tant de l'entorn com dels altres agents del sistema (Figura 2.1).

Figura 2.1: Exemple de SMA de tres agents



2.1.2. Enfocaments

Existeixen dos enfocaments per construir sistemes multi-agents:

- L'enfocament formal o clàssic, en què l'objectiu és dotar dels agents de la major intel·ligència possible utilitzant descripcions formals del problema a resoldre i de fer reposar el funcionament del sistema en aquestes capacitats cognitives.

- L'enfocament constructivista, que persegueix la idea de brindar intel·ligència al conjunt de tots els agents, per a que a través de mecanismes d'interacció, el mateix sistema genere comportament intel·ligent que no necessàriament estava planejat des d'un principi o definit. Aquest tipus de conducta és habitualment anomenat comportament emergent.

2.1.3. Característiques

- Autonomia: els agents són total o parcialment autònoms.

- Visió local: ningun agent té una visió global del sistema, o el sistema és massa complex per a un sol agent.
- Descentralització: no hi ha un agent de control designat (en eixe cas el sistema es reduiria a un sistema monolític).
- Capacitat d'inferència: l'habilitat d'un agent de treballar amb metes abstractes deduint una observació en generalitzar la informació que rep.
- Reactiu: l'agent detecta la condició de l'ambient i hi respon en un temps adequat.
- Pro-actiu: els agents no sempre han d'actuar només a causa dels estímuls de l'entorn, també han de decidir si actuar o realitzar alguna acció per aconseguir els seus objectius.
- Sociabilitat: un agent és capaç d'interactuar amb altres agents per assolir un objectiu. Comparteixen informació per ajudar-se amb altres agents a resoldre un problema.

2.1.4. Avantatges

Els sistemes multi-agent tenen distints avantatges davant altres aproximacions més centralitzades o individuals:

- Distribueixen recursos i capacitats computacionals a través d'una xarxa d'agents interconnectats. Mentre que un sistema centralitzat pot estar afectat per limitacions de recursos, un sistema multi-agent està descentralitzat i, per tant, és més tolerant a falles.
- Permeten la interconnexió i la interoperació de múltiples sistemes heretats existents.
- Ofereixen solucions en situacions en què l'experiència es distribueix espacial i temporalment.
- Milloren el rendiment global del sistema, específicament en les dimensions d'eficiència computacional, fiabilitat, extensibilitat, robustesa, manteniment, capacitat de resposta, flexibilitat i reutilització.

2.1.5. Plataformes per a desenvolupar agents

Per a desenvolupar agents i sistemes d'agents podem trobar distintes ferramentes o plataformes, la majoria basades en Python o Java. Podem veure plataformes com ara JADE, JASON, JADEX, PADE o SPADE. A continuació s'expliquen amb més profunditat cadascuna d'aquestes ferramentes.

Jason [4] és el primer intèrpret complet d'una versió molt millorada d'AgentSpeak, que inclou també la comunicació entre agents basada en actes de parla. En

l'àrea dels llenguatges de programació orientats a agents en particular, AgentSpeak ha sigut un dels llenguatges abstractes més influents basats en l'arquitectura BDI (Beliefs-Desires-Intentions) que és un dels enfocaments més coneguts per al desenvolupament dels agents cognitius. El tipus d'agents programats amb AgentSpeak de vegades s'anomenen sistemes de planificació reactiva. Existeixen diverses implementacions ad hoc de sistemes BDI, però una característica important d'AgentSpeak és el seu fonament teòric: és una implementació de la semàntica operativa, donada formalment al llenguatge AgentSpeak i a la majoria de les extensions disponibles a Jason. Una altra característica important de Jason en comparació amb altres sistemes d'agent BDI és que està implementat en Java (per tant, multi-plataforma) i està disponible de codi obert [1], distribuït sota GNU LGPL [2].

JADE (Java Agent Development framework) [5] és un programari per ajudar al desenvolupament d'aplicacions d'agents que compleix amb les especificacions FIPA 2000 [3] d'interoperabilitat als sistemes intel·ligents multi-agent. JADE és un projecte de codi obert, i el sistema complet es pot descarregar des de la pàgina d'inici de JADE[4]. L'arquitectura de comunicació JADE intenta oferir missatgeria flexible i eficient, escollint de manera transparent el millor transport disponible i aprofitant la tecnologia d'objectes distribuïts d'última generació integrada a l'entorn d'execució de Java. Tot i que apareix com una entitat única al món exterior, una plataforma d'agents JADE és en sí mateix un sistema distribuït, ja que es pot dividir en diversos *hosts* amb un d'ells actuant com a interfície on es col·loquen els agents. Un sistema JADE consta d'un o més contenidors d'agents, cadascun en una màquina virtual Java independent i ofereix suport per a l'entorn d'execució a alguns agents JADE. El sistema d'execució JADE intenta oferir serveis de missatgeria eficients i flexibles a les aplicacions dels usuaris.

Jadex [6] és un marc d'agents que admet capacitats de raonament fàcils d'utilitzar mitjançant l'explotació del model BDI combinant-lo amb tècniques d'enginyeria de programari d'última generació com XML i Java. El sistema Jadex té una arquitectura abstracta i s'integra a la plataforma JADE. Aquesta integració sorgeix de la necessitat d'una plataforma d'agents que admeta tant programari intermedi com raonament, l'enfocament escollit va ser basar-se en una plataforma de programari intermediari madura existent, que s'utilitza àmpliament. La plataforma JADE[5] se centra també a implementar el model de referència FIPA, proporcionant la infraestructura de comunicació i els serveis de plataforma necessaris, com ara la gestió d'agents, i un conjunt d'eines de desenvolupament i depuració. Deixa intencionadament oberts molts dels problemes dels conceptes d'agent intern, oferint un model senzill basat en tasques en què un desenvolupador pot realitzar qualsevol tipus de comportament de l'agent. Això fa que siga molt adequat com a base per establir un motor de raonament sobre el qual partir. La seua arquitectura és en gran mesura independent de la plataforma subjacent.

[1] http://jason.sourceforge.net/wp/

[2] https://www.gnu.org/licenses/lgpl-3.0.en.html

[3] Foundation for Intelligent Physical Agents, http://www.fipa.org/repository/fipa2000.html

[4] https://jade.tilab.com/

[5] https://jade.tilab.com/

Es basa en comportaments, determinats únicament per les creences, objectius i plans concrets dels agents.

PADE [7] és un marc per al desenvolupament, execució i gestió d'entorns de sistemes multi-agent de computació distribuïda. PADE està escrit 100% en llenguatge Python i utilitza les biblioteques Twisted[6] per implementar la comunicació entre els nodes de la xarxa. PADE és de codi obert[7] sota els termes de la llicència del MIT i està desenvolupat per Smart Grids Group (GREI) al Departament d'Enginyeria Elèctrica de la Universitat Federal de Ceará, Brasil. El marc PADE es va desenvolupar tenint en compte els sistemes d'automatització. Té interessants funcionalitats per al desenvolupament de sistemes multi-agent: l'abstracció d'orientació a objectes per a agents de construcció i els seus comportaments utilitzant conceptes d'orientació a objectes; un mòdul per crear i gestionar missatges FIPA-ACL; funcions de filtratge als missatges multi-agent i l'intercanvi d'objectes serialitzats.

SPADE [8] és una plataforma de sistemes multi-agent(SMA) de codi obert[8] que proporciona a l'usuari l'accés a entitats autònomes, proactives, intel·ligents i comunicatives mitjançant una interfície senzilla però potent. Un dels conceptes més rellevants de qualsevol SMA és el seu model d'agent. En el cas de SPADE, el seu model és similar als utilitzats en altres plataformes (com ara JADE). El model es basa internament en algunes abstraccions i mecanismes simples, començant pel mecanisme de connexió, mitjançant el qual cada agent es registra a SPADE utilitzant un identificador únic. Després de registrar-se, els agents poden crear un o diversos comportaments, que són tasques independents que executen les accions de l'agent. Aquesta és la plataforma en que es basa SimFleet, el simulador escollit per al treball, per la qual cosa més endavant s'explicarà amb major profunditat.

2.2 Context tecnològic

En quan al context tecnològic, s'han estudiat les diferents alternatives que hi havia abans d'elegir la definitiva per dur a terme el projecte. També s'han revisat treballs acadèmics similars per a comprovar les conclusions dels estudis més recents.

2.2.1. Simuladors

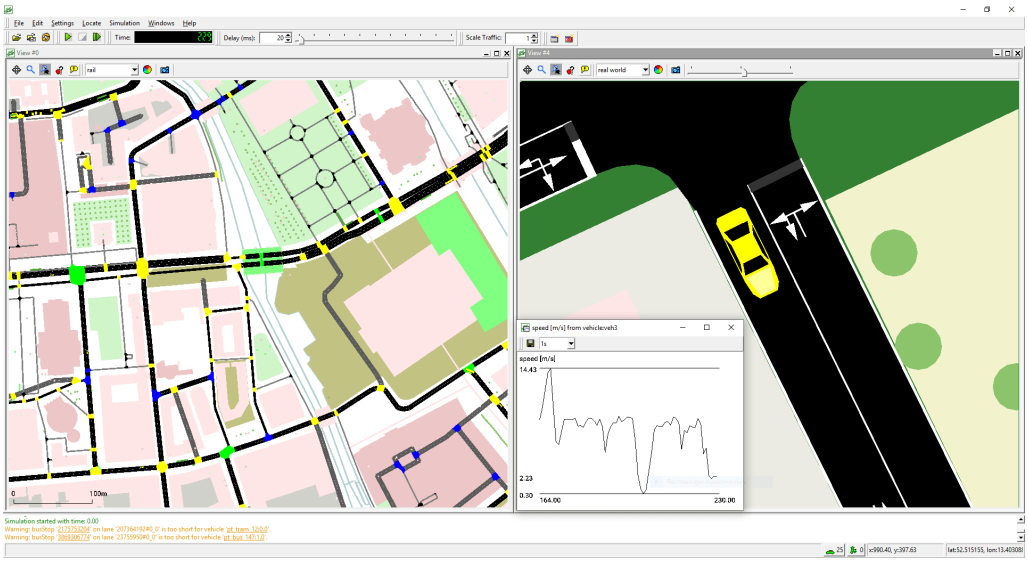
Els principals simuladors de transport urbà que s'han estudiat han sigut MATSim, SUMO, AnyLogic, Matisse, Vissim, Carla i per últim, SimFleet.

El primer, MATSim [9], proporciona un marc per implementar simulacions de transport basades en agents a gran escala, format per diversos mòduls que es poden combinar o utilitzar de manera independent, els quals es poden substituir per implementacions personalitzades.

[6] https://pypi.org/project/Twisted/
[7] https://github.com/grei-ufc/pade
[8] https://github.com/javipalanca/spade

SUMO [10] és un simulador de trànsit gratuït i de codi obert que permet modelar sistemes de trànsit intermodal, inclosos els vehicles de carretera, el transport públic i els vianants. SUMO inclou una gran quantitat d'eines de suport que automatitzen les tasques bàsiques per a la creació, l'execució i l'avaluació de simulacions de trànsit, com ara la importació de xarxa, el càlcul de rutes, la visualització i el càlcul d'emissions (Figura 2.2).

Figura 2.2: Simulador SUMO



AnyLogic [11] és una eina de modelatge de simulació multi-mètode desenvolupada per The AnyLogic Company, un dels simuladors amb més renom de la llista. Suporta metodologies de simulació basades en agents, esdeveniments discrets i dinàmiques del sistema. S'utilitza per simular: mercats i competència, sanitat, fabricació, cadenes de subministrament i logística, comerç minorista, processos empresarials, dinàmica social i d'ecosistemes, defensa, gestió de projectes i actius, informàtica, aeroespacial i per últim, el que ens interessa per al treball, dinàmica de vianants i trànsit urbà (Figura 2.3).

El simulador microscòpic MATISSE [12], per a sistemes de transport intel·ligents (ITS) basats en agents, és un software dissenyat a la University of Texas, Dallas. MATISSE ofereix classes abstractes per a la definició d'agents de vehicles i controladors d'intersecció, així una GUI de control (Figura 2.4), que permet a l'usuari canviar diverses propietats de l'agent en temps d'execució. El convertidor OSM importa xarxes de trànsit senceres des de l'Open Street Map[9].

Vissim [13], del grup TPV, simula interaccions complexes de vehicles de forma realista a nivell microscòpic (Figura 2.5). Modela la demanda, l'oferta i revisa el comportament de moviment amb detall. Aporta gran quantitat d'estadístiques sobre el flux, les emissions dels vehicles, el trànsit... És també un software conegut mundialment.

CARLA [14] té com a objectiu donar suport al desenvolupament, formació i validació de sistemes de conducció autònoma. A més del codi i protocols de codi obert, CARLA ofereix actius digitals oberts (dissenys urbans, edificis, vehicles) per a fer un ús lliure i gratuït d'ells. La plataforma de simulació està dissenyada

[9] https://www.openstreetmap.org/

Figura 2.3: Simulador AnyLogic

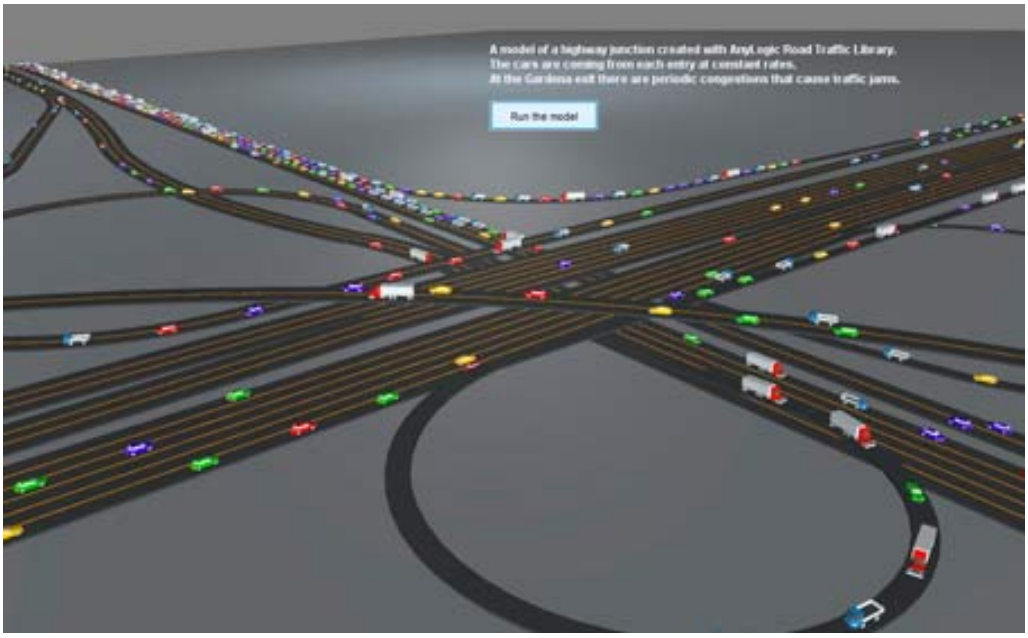
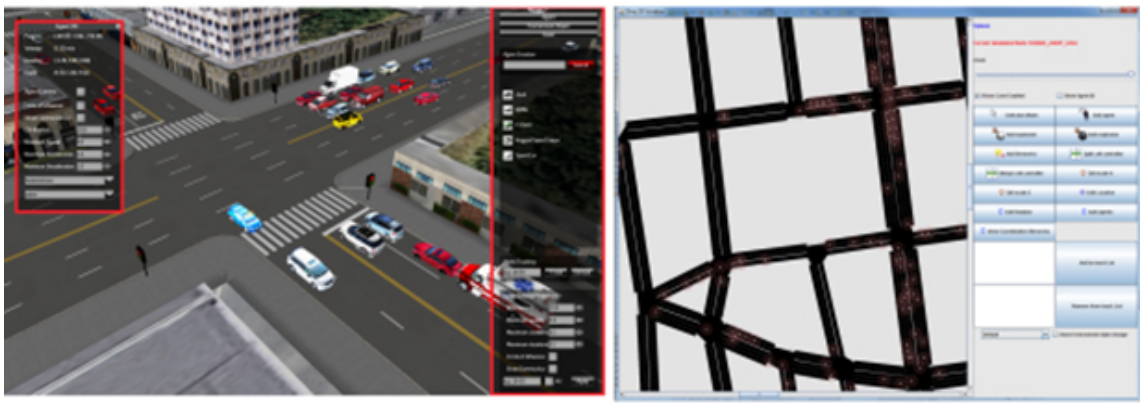


Figura 2.4: Simulador MATISSE



per a poder modificar sensors, condicions ambientals, control total de tots els actors estàtics i dinàmics o generació de mapes. És un simulador amb una interfície gràfica molt desenvolupada (Figura 2.6).

Finalment, s'ha decidit treballar amb SimFleet [15], un simulador de flotes urbanes basat en agents construït a la plataforma SPADE. Aquest simulador de flotes es va crear per permetre simulacions complexes sobre ciutats on un gran nombre d'agents interactuen per realitzar la coordinació global. Permet crear noves estratègies, la variació dels paràmetres i és de codi obert. Aquest últim punt, junt amb el fet de ser desenvolupat al grup d'investigació GTI-IA (Grup de Tecnologia Informàtica - Intel·ligència Artificial) de l'Institut Valencià d'Investigació en Intel·ligència Artificial (VRAIN) de la Universitat Politècnica de València, ha sigut el que ha conclòs l'elecció. Treballar en un entorn proper i familiar per als tutors és una avantatge a aprofitar. Tot i que té una de les interfícies gràfiques més simples dels simuladors mencionats (Figura 2.7), no és ningun problema ja que

Figura 2.5: Simulador Vissim

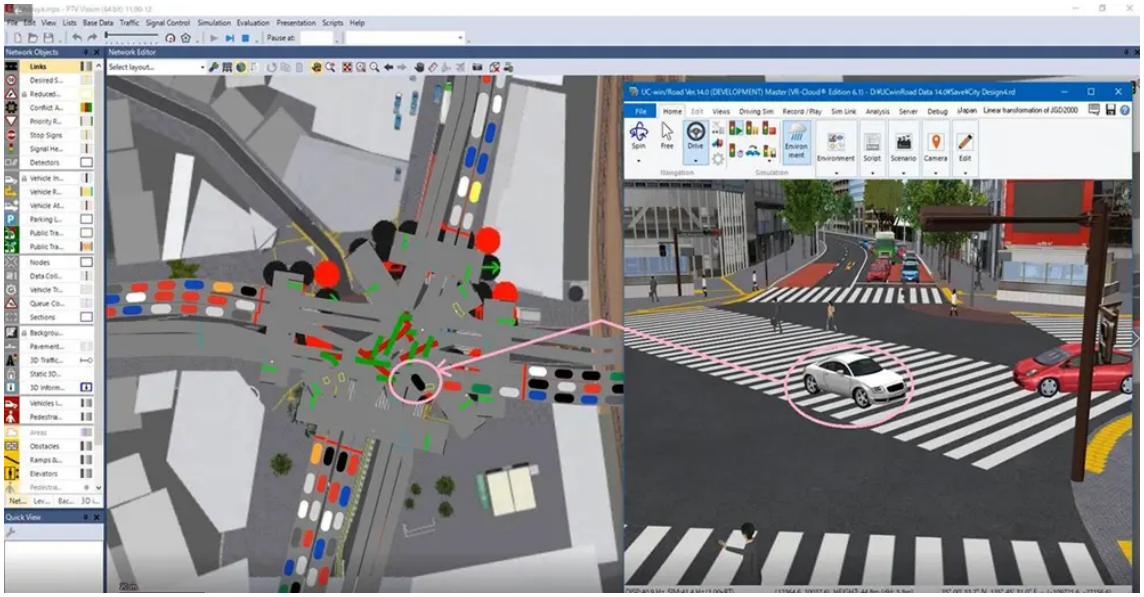


Figura 2.6: Simulador CARLA

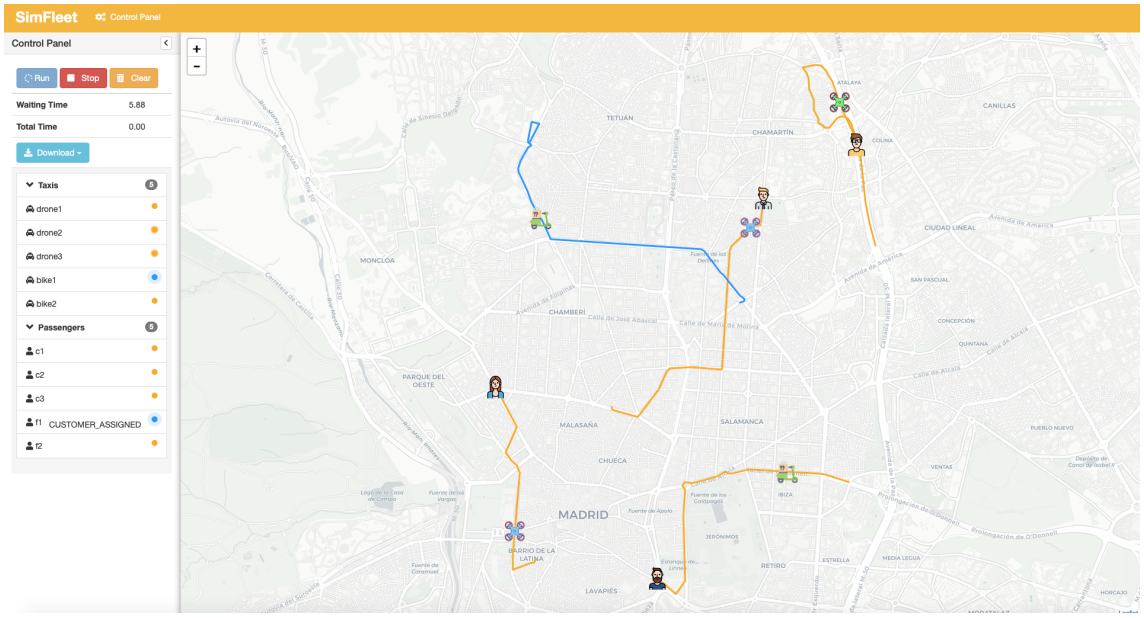
l'objectiu del projecte és aconseguir uns resultats estadístics de les simulacions i no la visualització d'aquestos.

2.2.2. Treballs relacionats

En relació al tema del treball, hi ha altres documents que arrepleguen resultats d'estudis similars. Saber quin tema tracten i com l'aborden (quines eines s'utilitzen, la metodologia, com es validen els resultats...) és molt interessant de cara a la realització del treball, per a poder adaptar idees útils o evitar pràctiques que no donen bons resultats.

El primer document [16] és un estudi mitjançant el simulador MATSim, en el qual es comprova les millors combinacions i distribucions de vehicles per al transport d'uns clients als que es busca optimitzar el servei oferit. S'explica també un generador de càrrega per al simulador, que bàsicament és un generador

Figura 2.7: Simulador SimFleet



d'agents al mapa, que es pot configurar segons el tipus de distribució desitjada. El resultat obtingut al fer simulacions a un mapa de la ciutat de València explica que la millor distribució és aleatòria o balancejada realitzant carsharing.

El segon article [17] tracta un tema similar, aconseguint modelitzar un sistema que realitza rutes totalment flexibles, parades sota demanda, una flota descentralitzada de vehicles interessats, cada vehicle amb capacitat per transportar de quatre a vuit passatgers, i ofereix reserves de viatges així com sol·licituds de viatge en temps real. A les conclusions, es detalla que no es necessita cap agent central per a fer tasques de comunicació entre els agents de transport. S'expliquen l'algoritme de subhasta de Bertsekas i el de coordinació per la millor resposta com a coordinadors de les operacions de la flota. Com a entorn es proposa utilitzar SimFleet també.

El tercer informe [18], se centra en la generació d'un escenari de simulació senzill a la configuració de MATSim. Es crea una població artificial d'agents per a Ille-de-France (regió de París), es modela el sistema de carreteres i transport públic de la regió i, finalment, es presenta un exemple de simulació en l'escenari. El procés d'aquest informe pretén ser un model bàsic per a la creació d'un escenari de simulació molt més detallat i basat en evidències. Mostrant quins passos s'han fet per a aquesta versió senzilla de l'escenari hauria de fer-se evident què és necessari per a una configuració més complexa.

En quart lloc tenim un article [19] on es presenta una anàlisi sobre simulació basada en agents. L'estudi, realitzat com a part d'un projecte alemany anomenat "elektroMobil", utilitza l'àrea metropolitana de la ciutat de Berlín (Alemanya) com a cas de prova: una ciutat amb múltiples operadors al mercat que ofereixen cotxes compartits tant a l'estació com a flotació lliure. S'han simulat tres escenaris diferents. L'escenari bàsic imita el sistema de transport real i suposa una oferta de cotxe compartit basada en l'estació. Dues simulacions es basen en una previsió de població per a anys futurs i s'assumeixen diversos canvis en l'oferta. En el

primer, es prova un carsharing més gran basat en estació, mentre que en el segon s'afegeix un gran carsharing flotant. Es mostra com es compara l'ús compartit de cotxes basat en l'estació i l'ús compartit lliure i els resultats poden ajudar a trobar estratègies per ampliar l'oferta de cotxe compartit a Berlín i com combinar l'ús compartit de cotxe lliure i l'estació. Aquesta és la idea més propera a la del present projecte, encara que utilitzant altres tecnologies.

Per últim, el treball [20] fa ús de SimFleet també, per a estudiar l'optimització del trànsit urbà. Per a resoldre el problema plantejat, utilitzen agents racionals amb interessos propis que per aconseguir unes simulacions més realistes. Implementen un mètode basat en teoria de jocs per aconseguir la comunicació entre agents; solució des de la qual cap agent te una altra alternativa millor. A més, les funcionalitats de SimFleet s'amplien amb la implementació de diversos mòduls que faciliten la creació d'escenaris de simulació complexos i introdueixen un nou tipus de flota urbana, fent-lo així un simulador més complet. És el projecte que més s'apropa al present treball en quan a l'ús del simulador.

CAPÍTOL 3
Solució proposada

En aquest capítol del treball es presenta la solució proposada d'una forma conceptual.

Primerament es descriurà la solució, definint en que consistirà a grans trets. A continuació es presentarà el pla de treball per a desenvolupar aquesta solució, amb unes estimacions temporals i una planificació a seguir.

3.1 En què consisteix

Davant del problema que es presenta, s'ha plantejat una solució basada en un sistema multi-agent. La idea principal és aconseguir dissenyar un sistema de transport *demand-responsive*, és a dir, que funciona per demanda. Ha de ser similar a un taxi i a un bus. Un vehicle pot arreplegar distints clients en un mateix viatge fins a una certa capacitat, com ocorre amb un bus, però també transporta a cada un dels clients del seu origen al seu destí concrets, com un taxi.

El sistema proposat realitzarà les simulacions a un mapa real, amb rutes existents, encara que sense tindre en compte el combustible, ja que es dona per suposat que és suficient per al temps que dura cada simulació. A una situació real, tant un autobús com un taxi tenen capacitat suficient al depòsit per a completar una jornada sense haver de recarregar el combustible. Tampoc es valoraran els semàfors ni ningun tipus de parades, ja que no són paràmetres controlables i no ens aporten informació rellevant per als nostres objectius.

La idea és aconseguir simulacions realistes amb aquest funcionament, per a poder comprovar l'ús de diferents capacitats màximes del vehicle i distintes estratègies. La idea de provar diverses capacitats és per a saber com afecta que puguen anar simultàniament més o menys clients al vehicle. Les estratègies consistiran en distints mètodes per a obtindre la ubicació a la qual viatjarà el transport en cada moment, bé l'origen d'un client o la destinació d'un dels passatgers. La idea es desenvolupar una serie d'estratègies on algunes siguen més bàsiques que altres, per a poder veure les diferències als resultats.

Els resultats a analitzar per a comparar les distintes estratègies i capacitats seran principalment temporals. Es pretenen analitzar els temps d'espera dels clients i el que tarden en arribar als seus destins, a banda del temps total de la simulació, el qual realment és el temps que es tarda en atendre a tots els clients.

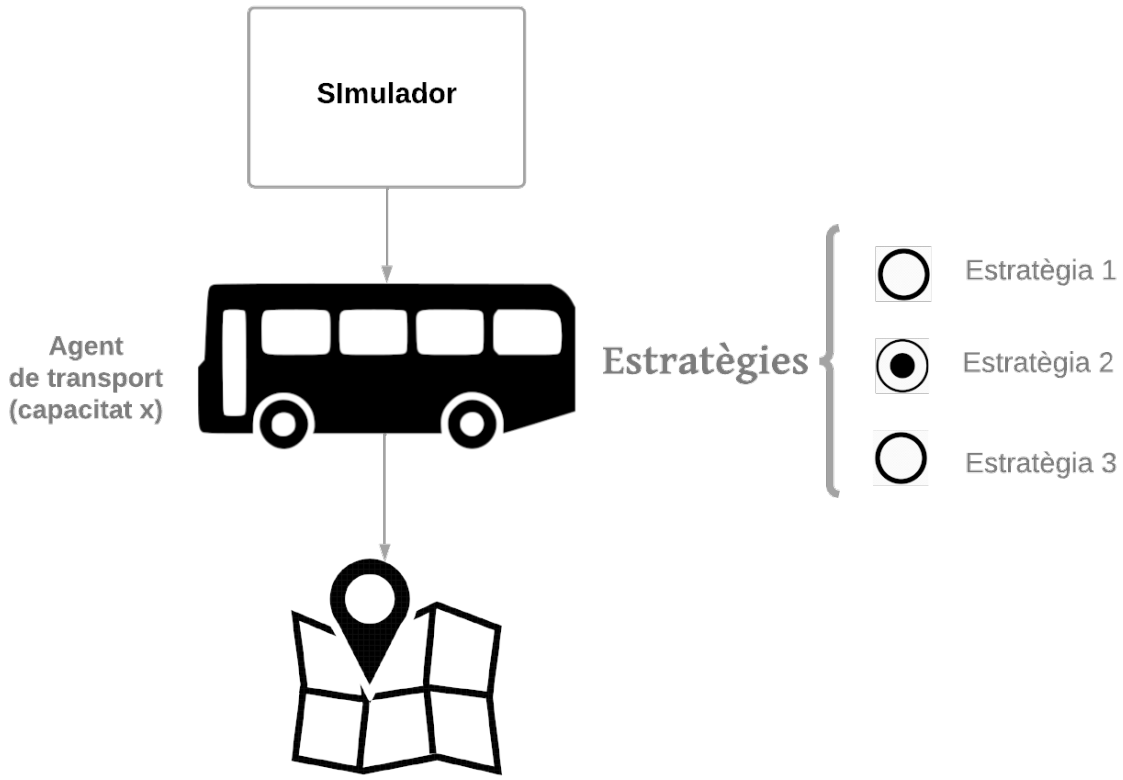


Figura 3.1: Esquema de la solució plantejada

També es compararan les distàncies recorregudes en les diferents proves, ja que serà més eficient l'estratègia que menys distància obligue a recórrer.

A la Figura 3.1 es mostra un esquema conceptual de la solució que s'ha plantejat, sense entrar en els detalls de quin simulador s'utilitzarà, ni de quines capacitats o estratègies es farà ús. Bàsicament, el vehicle funciona sobre un simulador amb una capacitat determinada de clients. El mapa o zona per la que circula el vehicle es pot elegir també, i a banda hi haurà una sèrie d'estratègies les quals es provaran per a determinar quina és la que millors resultats obté en les simulacions.

3.2 Pla de treball

El pla a seguir comença per la modificació o edició del software ja existent. Es vol aconseguir modificar l'agent transport per a que siga capaç de transportar a més d'un client simultàniament. Per a aquest primer pas s'estima un duració d'entre un i dos mesos, on s'inclou entendre el funcionament del sistema, la seua estructura, com realitzar simulacions i canviar els paràmetres, i llegir documentació.

En segon lloc, una vegada hi haja un control sobre el simulador a nivell usuari, la intenció es modificar el codi necessari de simulador per a permetre que el vehicle puga portar a dos o més clients a un mateix viatge. Posteriorment, s'haurà de plantejar i desenvolupar diferents estratègies per a comprovar quina resulta

més eficient, i en quins casos. Per a aquest punt s'estimen dos mesos, per els problemes que puguen sorgir, ja que és la fase més delicada en quan a possibilitat d'errors.

Finalment les proves experimentals i les conclusions tenen una estimació d'un mes, on el més complicat és plantejar una serie de proves amb certa lògica per a obtindre uns resultats significatius.

CAPÍTOL 4
Disseny de la solució

En aquesta part del treball es presentarà com s'ha dissenyat la solució proposada anteriorment.

Primerament es mostrarà SimFleet, el simulador elegit per a desenvolupar la solució. Tot i que ja s'havia presentat breument a la secció 2.2.1, s'explicarà en que consisteix de forma més detallada, junt amb SPADE, la plataforma sobre la que funciona SimFleet. Seguidament es descriurà l'arquitectura d'aquest simulador, i per últim, un generador de càrrega per a SimFleet amb el que es pretén generar uns fitxers de proves per a la realització dels experiments.

4.1 Tecnologia utilitzada: SimFleet

Com s'ha explicat al capítol anterior, SimFleet és el simulador elegit per a realitzar aquest treball. A continuació s'explicarà al detall en que consisteix i com funciona, i també una descripció de SPADE, la plataforma de sistemes multi-agent sobre la que es basa SimFleet [15].

4.1.1. Descripció

SimFleet és un simulador de flotes de sistemes multi-agent per a provar estratègies. L'objectiu de SimFleet és simular flotes obertes de vehicles i desenvolupar algoritmes intel·ligents de coordinació i negociació per gestionar diferents situacions amb flotes obertes. SimFleet està construït sobre un sistema multi-agent anomenat SPADE. El simulador es compon de tres tipus d'agents: conductors, clients i gestors de flotes.

L'eina proporciona una interfície de desenvolupament que té una interfície gràfica (GUI) per mostrar la simulació visual, la interfície d'execució de la línia d'ordres i un back-end (API) on és fàcil introduir nous comportaments per a ser provats.

La GUI és la part de l'aplicació que executa el simulador on es mostren els vehicles de transport i els elements a transportar dins del mapa de la ciutat escollit, i on es pot observar l'evolució de la simulació; aquesta interfície inclou un conjunt limitat d'interaccions (inici i aturada de la simulació, zoom, col·locació aleatòria de nous vehicles i elements, estadístiques de simulació, etc.).

La interfície del *command line* és la manera d'iniciar simulacions fora de línia i inclou totes les configuracions possibles disponibles a la simulació; aquesta és una manera molt còmoda d'ampliar la funcionalitat de l'eina de simulació (en general, o per a un entorn de flota particular) sense haver de modificar la interfície gràfica. Ens permet simular escenaris particulars amb ubicació inicial fixa per a vehicles o elements, així com establir les estratègies de negociació per als agents de simulació; d'aquesta manera, la simulació pot comparar de manera consistent diferents esquemes de negociació en igualtat de condicions.

La tercera interfície és l'API per implementar estratègies de negociació personalitzades; algunes de les funcions de l'API són generals, però moltes altres són específiques per a entorns de flotes particulars (entrega de paquets, servei de taxi, etc.).

4.1.2. SPADE

Com s'ha introduït abans, SPADE [8] és una plataforma de sistemes multi-agent (SMA) que proporciona a l'usuari l'accés a entitats autònomes, proactives, intel·ligents i comunicatives mitjançant una interfície senzilla però potent, motiu pel qual va ser escollit per al desenvolupament de SimFleet. Es tracta d'una plataforma i llibreria d'agents que acompleix les especificacions FIPA[1]. És un software multi-plataforma i multi-llenguatge, és a dir, pot donar servici a agents desenvolupats en diversos llenguatges de programació.

Un dels conceptes més rellevants de qualsevol SMA és el seu model d'agent. En el cas de SPADE, el seu és similar als utilitzats en altres plataformes (com ara Jade). El model es basa internament en algunes abstraccions i mecanismes simples, començant pel mecanisme de connexió, mitjançant el qual cada agent es registra a SPADE mitjançant un identificador únic (el format és "nom d'usuari@servidor") i una contrasenya. Després de registrar-se, els agents poden crear un o diversos comportaments, que són tasques independents que executen les accions de l'agent. Els comportaments poden ser de diversos tipus, i cada tipus produeix un patró d'execució particular dissenyat per donar suport a un requisit d'execució típic dels agents en un sistema multi-agent. En particular, SPADE admet cinc tipus de comportament: Cíclic, One-Shot, Periòdic, Time-Out i Finite State Machine. El tercer mecanisme principal és el despatxador de missatges que SPADE associa amb cada agent registrat. Aquest component actua com a carter, redirigeix qualsevol missatge entrant per a l'agent al comportament o comportaments particulars que poden estar esperant el missatge i transmetent els missatges sortints de qualsevol comportament al sistema de comunicació de SPADE.

Entre les tecnologies, SPADE es basa en la selecció d'un protocol de comunicació particular com a component primordial del sistema multi-agent, ja que pot aportar funcions molt valuoses a entitats socials intel·ligents, autònomes, com ara agents. En particular, SPADE incorpora XMPP[2] (eXtensible Messaging and Presence Protocol), que és un protocol obert per a missatgeria instantània i notificació de presència. XMPP és un protocol basat en XML que permet a les entitats intercanviar missatges (amb aquestes entitats són humans, agents, artefactes, etc.) i

[1] http://www.fipa.org
[2] https://xmpp.org/

que també proporciona un mecanisme de notificació de presència pel qual qualsevol entitat pot tenir una llista d'altres entitats com a contactes, i ser notificat quan algun d'aquests contactes canvia el seu estat (p. ex., quan un contacte està connectat o desconnectat, quan està ocupat, etc.). A més, XMPP es defineix com un protocol extensible, és a dir, moltes de les seves característiques es proposen com a extensions (anomenades XEP[3]); actualment ofereix nombroses extensions per a diferents finalitats, i està obert a propostes de la comunitat per tal de flexibilitzar i fer el protocol més útil. Per això, XMPP és considerat com el protocol estàndard universal per a la missatgeria instantània per part d'entitats com l'[4] o el W3C[5], i té un ús molt estès a la indústria. Whatsapp, Google Talk, Facebook Messenger o iMessage d'Apple són alguns exemples d'aplicacions que utilitzen XMPP, o alguna variant d'aquest protocol.

Els agents SPADE també tenen una interfície basada en web (Figura 4.1) per crear interfícies d'aplicacions personalitzades per als agents, que SimFleet també utilitza per mostrar com cada agent es mou per la ciutat en un mapa que representa totes les rutes que prenen els agents.

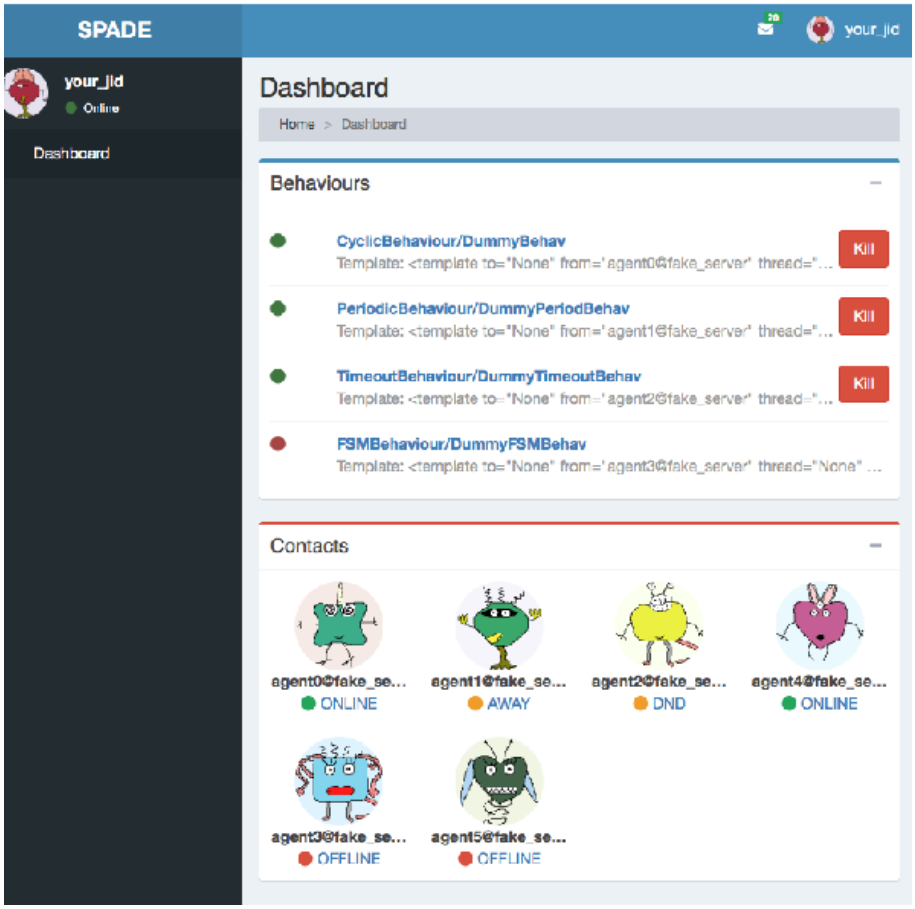


Figura 4.1: Interfície web de SPADE, on es mostra un agent amb els seus comportaments i contactes

[3]XMPP Extension Protocols, https://xmpp.org/extensions/xep-0001.html
[4]Internet Engineering Task Force, https://www.ietf.org
[5]World Wide Web Consortium, w3.org

4.1.3. Arquitectura

Internament SimFleet s'estructura en quatre capes, dissenyades per facilitar l'adaptació a escenaris de negociació i flota particulars. A la figura 4.2 es representa l'arquitectura del simulador, la flota, els agents i les seues respectives estratègies.

El simulador és l'agent que controla el procés de simulació i serveix la GUI, i està configurat per localitzar la pantalla de simulació en una ciutat concreta, que es pot modificar. A més del mapa de la ciutat, el simulador es basa principalment en un agent de planificació de rutes que calcula rutes vàlides pels carrers de la ciutat per als vehicles que viatgen d'un punt a un altre de la ciutat. El simulador també s'encarrega de crear la resta d'agents d'aplicació en la simulació.

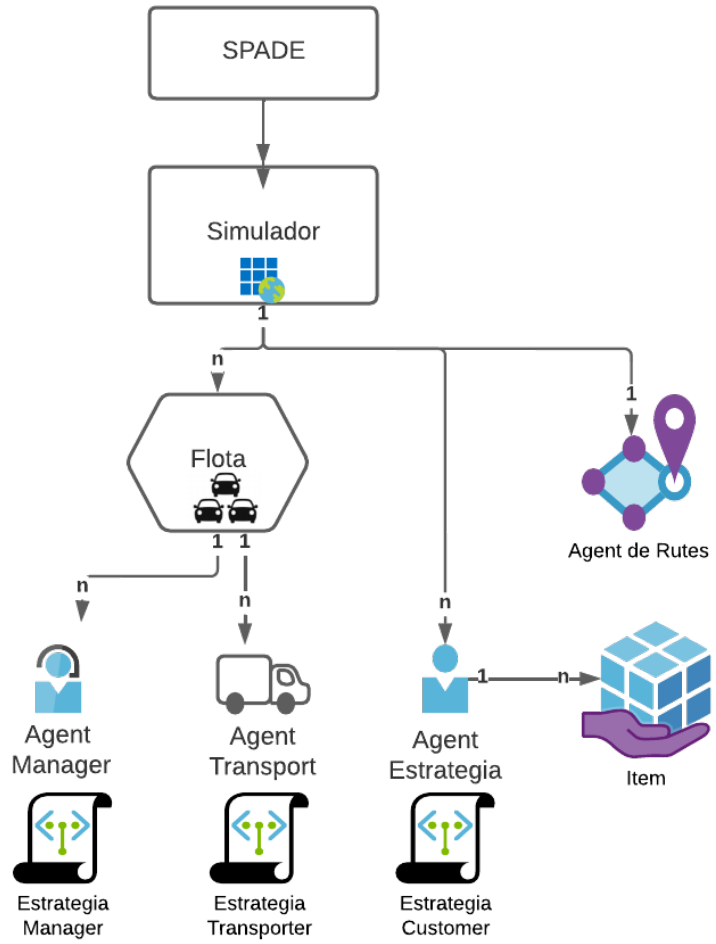
Una flota representa un grup de vehicles capaços de transportar mercaderies o persones d'un punt a un altre de la ciutat. Cada vehicle de la flota té el seu propi agent a la simulació, anomenat Transport.

A més dels agents transportistes, cada flota inclou un altre agent, anomenat Manager o Gestor de Flota, que s'encarrega de rebre les peticions dels clients i organitzar-les. A la capa d'agent, l'eina inclou un agent per al Gestor de Flotes i tots els Transportistes de cada flota, i també un agent per a cada Client que puga emetre peticions de transport a la simulació. A l'agent de transport es pot modificar la velocitat màxima, cost per quilòmetre, la flota, el nom del gestor de la flota, la posició inicial i la seua icona al simulador. D'altra banda, un client és un agent que pot sol·licitar serveis de transport per a qualsevol flota disponible, i que inclou alguns atributs com la quantitat de diners disponible, la llista d'articles a transportar i la llista dels noms de tots els gestors de flotes disponibles.

La quarta capa és el conjunt d'estratègies de negociació que apliquen els agents de simulació per sol·licitar, seleccionar, organitzar i coordinar els serveis de transport. Cada agent incorpora la seua pròpia estratègia segons el seu rol en l'entorn de simulació (Client, Transportador o Gestor de Flotes) i els seus objectius particulars.

L'arquitectura SimFleet proporciona una API ben definida per tal d'incorporar aquestes estratègies als agents. En particular, cada agent incorpora un comportament SPADE específic que defineix els objectius de negociació i les interaccions amb els agents. A més, aquest comportament s'estructura internament com una màquina d'estats finits (heretada de la classe SimFleet FSMStrategyBehaviour) [21] on cada estat representa una situació possible en l'estratègia de negociació de l'agent (i implementa les accions a executar en aquesta situació), i cada transició representa un possible canvi d'un estat a un altre dins l'estratègia. Aquesta capa s'ha dissenyat utilitzant l'anomenat Strategy Pattern, que permet incorporar i seleccionar per a l'execució diferents versions alternatives d'un algoritme. SimFleet utilitza aquest patró de disseny per incorporar dinàmicament nous comportaments de negociació als agents de simulació en temps d'execució. Els fitxers que contenen el codi d'aquests comportaments es poden especificar a la línia d'ordres quan es fa una simulació, pel que no cal modificar els fitxers de codi de l'eina.

Figura 4.2: Arquitectura de SimFleet



4.1.4. Generador de càrrega

Per a realitzar els experiments al simulador i extraure unes conclusions contrastades, s'ha fet ús d'un generador de càrrega de flotes, que s'explica amb més profunditat al treball [22]. Aquest inclou un generador d'estacions de recàrrega, que omple l'àrea de simulació amb un nombre determinat d'estacions de recàrrega seguint un patró determinat; i un generador de càrrega de tots tipus d'agents. Al nostre cas en particular sols s'ha fet ús del segon generador.

Una simulació a SimFleet es defineix pel seu fitxer de configuració. Crear experiments automàticament significa generar nous fitxers de configuració o omplir-ne un d'anterior amb dades segons alguns paràmetres. Per a això, tots els generadors inclouen l'opció d'obtindre com a entrada un fitxer de configuració. Cada generador treballa amb un fitxer GeoJSON que defineix l'àrea del món real on es farà la nostra simulació i, per tant, s'han de poblar. Aquestes simulacions es solen realitzar dins dels límits d'una ciutat, pel que podem dir que treballem sobre el mapa d'una ciutat. Com que representa una ubicació del món real, totes les geometries definides al mapa s'indiquen amb punts de latitud-longitud; per manipular-los s'utilitza la biblioteca Python Shapely4[6].

[6] https://pypi.org/project/Shapely/

Entrant més en detall del generador de càrrega, s'utilitza per crear una càrrega aleatòria o informada a la simulació. Aquesta càrrega es pot adaptar a qualsevol dels tipus d'agent que ofereix SimFleet: vehicles elèctrics, flotes de taxis, clients de taxis, vehicles de lliurament, paquets, etc. Els paràmetres rellevants d'aquest generador són: tipus d'agent t, quantitat d'agents n, distància mínima *minDist* en metres, retard d'inici d en segons i quantitat d'agents per lot *agentsPerBatch*. El generador pretén crear un moviment d'almenys la distància mínima *minDist* en metres de n agents de tipus t dins dels límits d'un mapa de ciutat determinat. El paràmetre de retard d determina en quin moment de la simulació començaran a executar-se els agents generats; per defecte, és 0. La quantitat d'agents per lot s'introdueix per donar diferents retards als conjunts d'agents per lot d'agents, que començaran la seua execució al mateix temps. Això pot ser útil quan es genera un gran nombre d'agents i permet l'ús del generador de càrrega per introduir, en una mateixa simulació, diferents tipus d'agents en diverses quantitats, amb diferents retards i mides de lot, per crear un sistema complex.

El generador, com ja s'ha esmentat abans, es pot configurar per a ser de moviment aleatori o informat.

La càrrega aleatòria es crea escollint una ruta aleatòria amb els punts d'origen com els de destinació aleatoris, que han de ser punts vàlids de l'àrea, i han d'estar almenys una distància mínima entre ells. Aquest procés es repeteix per crear n agents de tipus t. El punt d'origen determinarà on es generarà l'agent, mentre que el punt de destinació indica on acabarà la seua execució. Si el tipus d'agent és client o paquet, el moviment el realitza el vehicle de transport corresponent que el porta després de ser recollit.

D'altra banda, la versió informada del generador de càrrega pretén reproduir moviments més realistes al voltant del mapa de la ciutat. Per a això, cal facilitar al generador les dades rellevants a partir de les quals poder crear les rutes dels agents. Aquestes dades es poden obtenir de diverses fonts, com les plataformes de dades obertes que el govern d'una ciutat o país fa accessibles per als seus ciutadans. Aquest generador utilitza en concret informació de població, informació de trànsit i activitat de Twitter (quantitat de tuits geolocalitzats, de la xarxa social Twitter, tuitejats des d'una ubicació determinada). Per al nostre cas s'ha fet ús, com ja es detallarà més endavant, de la versió aleatòria.

CAPÍTOL 5
Implantació i desenvolupament

En aquest capítol es presenta la part més important del projecte, la del desenvolupament de la solució dissenyada.

Per a començar es mostraran les modificacions realitzades al codi previ de SimFleet. Després, relacionat amb l'anterior secció, es presentaran els comportaments amb que funcionarà la nova versió del simulador. Després d'aquest punt es descriuran també les estratègies elaborades.

Finalment es descriu com seran els fitxers de configuració per a les proves, amb els seus paràmetres i variables definides.

5.1 Modificació del codi

Una vegada plantejada la idea i les eines necessàries, s'han modificat les parts del codi font de SimFleet que s'han considerat rellevants. Els fitxers editats han sigut principalment el de l'agent de transport (*transport.py*) i el nou fitxer d'estratègies que hereta de la classe FSMStrategyBehaviour (*bus_strategy.py*). Per a definir els nous estats també s'ha hagut de modificar el fitxer on es declaren (*utils.py*). Els fitxers dels agents client i gestor de flota no ha sigut necessari editar-los ja que no afecten al nou funcionament.

En quan al fitxer agent de transport, s'han declarat nous atributs com la capacitat màxima de clients al vehicle, una cua de clients, la tasca actual i una llista dels passatgers del vehicle. El fitxer d'estratègies inclou els estats en els quals treballa l'agent de transport, amb la seua declaració i transicions.

5.2 Comportaments

Els comportaments o *behaviours* son una de les principals característiques en què es basa SimFleet. Defineixen com es comporta determinat agent davant les percepcions d'altres agents o de l'entorn, en forma de màquina d'estats. Una màquina d'estats és un conjunt d'estats amb connexions entre sí definides pel tipus d'esdeveniment que ocorre a l'entorn.

En aquest treball, la principal novetat és l'ús de dues màquines d'estat diferents. La primera d'elles és la que s'encarrega de la gestió de clients, i la segona s'encarrega de l'estratègia d'elecció de tasques. De forma anàloga podríem pensar en el primer comportament com un encarregat al vehicle per a acceptar o rebutjar les peticions dels clients, mentre que el segon seria el propi conductor, és a dir, qui s'encarrega dels viatges i l'ordre de les parades.

Gestió de clients

Aquest comportament ve des de l'anterior versió de SimFleet. Era l'únic que s'utilitzava per a la gestió dels clients, ja que sols podia haver un client com a màxim al transport en cada moment, per la qual cosa no era necessari un comportament per elegir el client.

En aquest *behaviour* es gestiona l'acceptació o rebuig a les sol·licituds dels clients. Per a ser acceptats, els clients han d'enviar una proposta, que el transport acceptarà si la capacitat màxima no està completada. En eixe cas, el transport enviarà una altra proposta al client, que en cas d'acceptar-la (pot donar-se el cas que accepte la proposta d'un altre transport abans), afegirà aquest client a la cua de tasques.

A continuació es mostra un extracte del codi necessari per a aquest comportament. Com es pot veure, és una classe que hereta de CyclicBehaviour, ja que és un comportament que es repeteix de forma cíclica per estar sempre escoltant noves propostes.

```
class ClientsManagementBehaviour(CyclicBehaviour):
    async def run(self):
        try:
            msg = await self.receive(timeout=60)
            if not msg:
                return
            logger.debug("Transport {} received: {}".format(self.agent.
                jid, msg.body))
            content = json.loads(msg.body)
            performative = msg.get_metadata("performative")
            if performative == REQUEST_PERFORMATIVE:
                if self.agent.current_capacity < self.agent.
                    max_capacity:
                    await self.send_proposal(content["customer_id"],
                        {})  # se envia proposal al client

            elif performative == ACCEPT_PERFORMATIVE:
                pick_time = time.time()
                self.agent.clients_queue[content["customer_id"]] = {'
                    pos': content["origin"], 'dest': content["dest"],'
                    task': 'pick_up', 'time' : pick_time}# task is
                    initially pick_up

            elif performative == REFUSE_PERFORMATIVE:
        except CancelledError:
            logger.debug("Cancelling async tasks...")
        except Exception as e:
            logger.error("EXCEPTION in Customer Management Behaviour
                {}: {}".format(self.agent.name, e))
```

Gestió de les tasques

La màquina d'estats per a la gestió de les tasques, és a dir, dels clients prèviament acceptats, està formada per set estats. La declaració d'aquests estats, que es poden veure a la figura 5.1, s'ha hagut de crear quasi des de zero, ja que sols s'ha mantingut des de l'anterior versió l'estat d'espera (*TRANSPORT WAITING*).

Aquest estat d'espera és l'estat d'inici i el base des d'on es selecciona l'acció a realitzar en cada moment. Aquesta acció, o tasca, serà una o altra depenent de l'estratègia que s'haja utilitzat. Si la tasca és recollir a un client, s'eliminarà aquest client de la cua i es passarà a l'estat de recollir a un client (*TRANSPORT GOING TO PICK UP*), després de cridar al mètode per arreplegar al futur passatger. En cas d'elegir deixar un client a la seua destinació, es farà la transició a l'estat de deixar clients (*TRANSPORT GOING TO DROP-OFF*).

L'estat al que es passa per a arreplegar a un client simplement és un estat auxiliar que canvia a l'estat on el transport es dirigeix cap al client (*TRANSPORT TO CUSTOMER*). A aquest estat es llança un callback per a detectar quan finalitza el viatge fins al client. En aquest punt, es torna a l'estat base d'espera per a elegir la nova tasca.

En quan a l'estat de deixar a un passatger, funciona de forma pareguda als dos estats de recollir al client. En aquest estat es llança l'ordre a l'agent de transport de moure's a la destinació, i es passa a l'estat *TRANSPORT TO DESTINATION*. Aquest segon estat també activarà un callback per a saber quan acaba el moviment del vehicle. De la mateixa forma que al cas anterior, es canvia a l'estat d'espera al acabar la tasca.

5.3 Estratègies desenvolupades

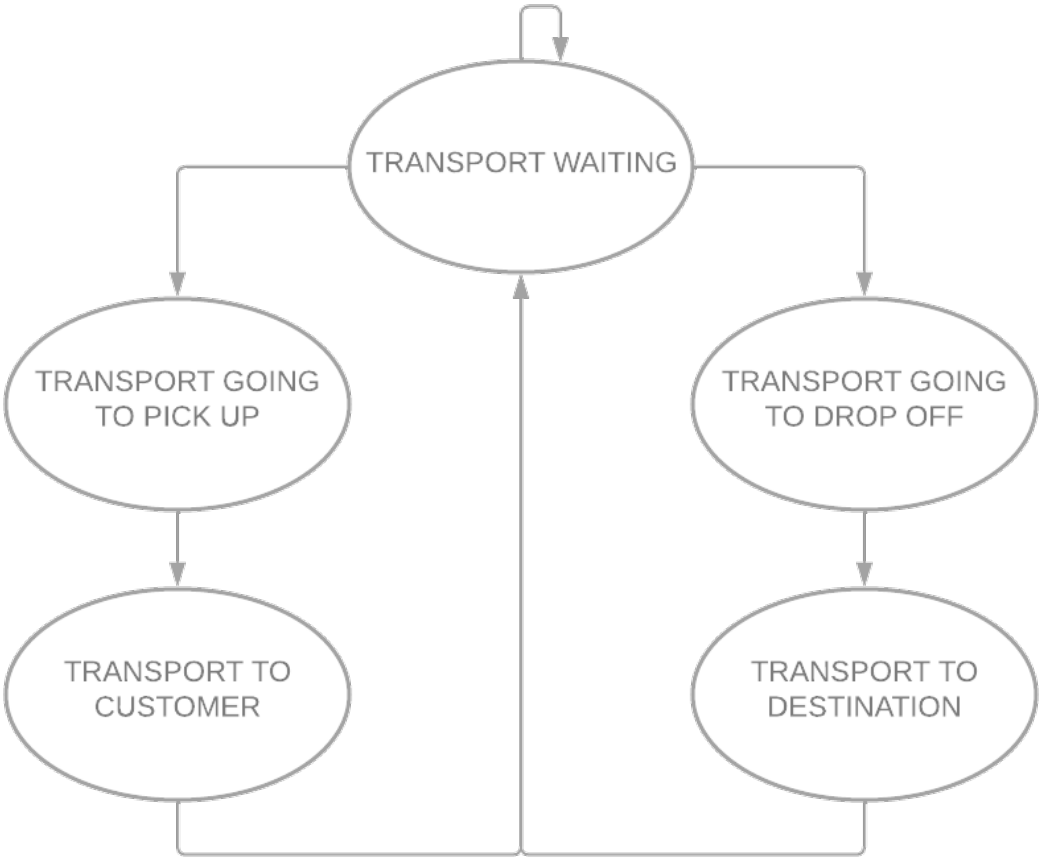
En total s'han creat quatre estratègies diferents per a triar la següent tasca a realitzar per el transport. Amb tasca es fa referència a arreplegar un client o deixar a un que ja està al transport al seu destí.

Les estratègies són les següents:

1. **FIFO**: s'ha anomenat així per l'ordre d'elecció de les tasques, ja que atén a cada client per ordre de arribada (First In First Out). És la més bàsica i no aporta ningun funcionament addicional a la versió anterior de SimFleet. Quan s'arreplega a un client, la següent tasca és deixar-lo al seu destí. Per aquesta raó, no hi haurà dos clients al transport en ningun moment. Els resultats d'aquesta estratègia simulen el funcionament bàsic del transport, per a veure clarament les diferències amb la resta d'estratègies.

2. **Proximitat** o **Proximitat Absoluta**: l'objectiu d'aquesta tàctica és atendre la tasca més propera en cada moment. Tenint en compte tant els destins dels clients que són al transport com els punts de recollida dels client que esperen a la ciutat. Es calcula la distància en metres entre la ubicació del vehicle i les posicions dels clients, per a extraure la tasca per a la qual menys distància s'haja que recórrer.

Figura 5.1: Transicions entre els estats definits



3. **Proximitat Relativa** o **Distància Balancejada**: es pot considerar una versió de l'anterior, ja que utilitza el mateix comportament, però redueix les distàncies dels clients que ja estan al transport (dona prioritat), per reduir el temps d'espera dins del vehicle. Es calcula un multiplicador entre 1 i una aproximació a 0, per a ponderar la distància i així obtindre una distància menor quan més temps estiga el client al transport. Tècnicament el valor que es compara ja no és la distància com a tal, sinó una xifra que expressa la prioritat; quan menor siga, major prioritat. S'estableixen uns límits per als valors del multiplicador, que van per 6, 8, 10, 12 i 13 minuts al vehicle, disminuint el multiplicador de 0.95 a 0.9, 0.7, 0.4 i cinc dividint pel temps al bus a partir dels 13 minuts. Amb açò es vol donar més prioritat quan més temps es porta al transport, reduint el multiplicador més cada vegada per a evitar que el client passe massa temps viatjant. Al llarg del treball es farà referència a aquesta estratègia amb les seues sigles, DB.

   Seguidament es mostra el fragment de codi on es realitza aquesta estratègia. Com es pot observar, *speed_ratio* és la variable encarregada d'ajustar els valors de temps a la velocitat per a que siga en proporció a una velocitat de 40 kilòmetres per hora.

```
1     def select_distance_balanced(self):# select with a balanced
          priority with time in bus(for passengers) and distance
```

```
        current_time = time.time()
        bus_pos = self.agent.get("current_pos")
        list_tasks = self.agent.clients_queue.copy()  # merge list
            of clients queue and passengers
        list_tasks.update(self.agent.passengers)
        min_dist = math.inf  # we set as initial minimum distance
           the largest number (infinite)
        for client in list_tasks:
            speed = self.agent.get("speed_in_kmh")
            speed_ratio = speed/40 #the ratio of the speed with a
               speed of 40kmph (realistic approach)
            penalty = 1
            if list_tasks[client]['task'] == 'pick_up':
                # check wether we have picked up the client or not
                    , to calculate with origin or destination
                dist = distance_in_meters(bus_pos,list_tasks[
                    client]['pos'])
            else:
                dist = distance_in_meters(bus_pos, list_tasks[
                    client]['dest'])
                time_in_bus = current_time - list_tasks[client][
                    'time']  # total time of a customer waiting in
                        the bus
                time_in_bus = (time_in_bus*speed_ratio)/60 #
                    proportional time if we were going with a 40
                    kmph speed, in minutes
                if time_in_bus > 6 and time_in_bus < 8: # if the
                    proportional waiting time is more than 6
                    minutes but lees than 8
                    penalty = 0.95
                elif time_in_bus < 10:
                    penalty = 0.9
                elif time_in_bus < 12:
                    penalty = 0.7
                elif time_in_bus < 13:
                    penalty = 0.4
                else:
                    penalty = 5 / time_in_bus  # will be a penalty
                        of 0.38 or less
            dist = dist * penalty
            if dist < min_dist:
                min_dist = dist
                next = client
        if list_tasks[next]['task'] == 'pick_up':
            task=(next,self.agent.clients_queue.pop(next))
        else:
            task = (next, self.agent.passengers[next])
        return task
```

Listing 5.1: Codi de l'estratègia DB

4. **TSP**: aquesta última tàctica utilitza l'algorisme TSP (Travel Salesman Problem) [23]. El problema del viatjant tracta de resoldre la problemàtica de visitar els distints punts d'un mapa o graf una sola vegada i recorrent la mínima distància possible, acabant al mateix punt d'inici. En el codi s'utilitza l'algoritme sols per a obtindre el primer punt que visitar per a aconseguir aquest recorregut òptim en quant a distància, i es recalcula per a elegir la

Algorithm 1: Dynamic Approach for TSP

Data: s: starting point; N: a subset of input cities; $dist()$: distance among the cities
Result: $Cost$: TSP result
$Visited[N] = 0$;
$Cost = 0$;
Procedure TSP(N, s)
  $Visited[s] = 1$;
  if $|N| = 2$ *and* $k \neq s$ **then**
    $Cost(N, k) = dist(s, k)$;
    Return Cost;
  else
    for $j \in N$ **do**
      for $i \in N$ *and* $visited[i] = 0$ **do**
        if $j \neq i$ *and* $j \neq s$ **then**
          $Cost(N, j) = \min (TSP(N - \{i\}, j) + dist(j, i))$
          $Visited[j] = 1$;
        end
      end
    end
  end
  Return $Cost$;
end

Figura 5.2: Pseudocodi de l'algoritme TSP [1]

tasca cada vegada. A priori pot parèixer molt similar a la Proximitat Absoluta, però no sempre el punt més proper ha de ser el primer per iniciar una ruta mínima al llarg d'una serie de punts. A la figura 5.2 es pot observar el pseudocodi de l'algoritme.

5.4 Fitxers de configuració per a les proves

La possibilitat de carregar escenaris a SimFleet ens permet repetir el mateix experiment tantes vegades com vulguem amb les mateixes condicions inicials. SimFleet admet carregar un fitxer de configuració que defineix tots els camps que es necessiten per carregar la mateixa informació repetidament. Un fitxer d'escenari s'ha de codificar en format JSON.

Els camps més importants que ha d'incloure el fitxer d'escenari són una llista de clients i una llista de transports.

Per als clients figuren les coordenades d'origen i destinació de cada un i el nom, i per al transport es defineixen la velocitat, el nom, l'autonomia i la posició inicial, com a principals paràmetres. Per al mànager es defineixen el nom i la ubicació inicial. També s'inclouen a tots els agents una contrasenya, el tipus de flota i una icona per a la interfície gràfica.

A continuació es mostra un exemple d'aquesta llista d'agents:

```
1
2     "fleets": [
3         {
```

```
            "password": "secret",
            "name": "fleetm1",
            "fleet_type": "taxi",
        }
    ],

    "transports": [
        {
            "speed": 2000,
            "fleet": "transport1@localhost",
            "fleet_type": "drone",
            "position": [40.41192762537371, -3.7105464935302734],
            "password": "secret",
            "name": "t1"
        }
    ],

    "customers": [
        {
            "destination": [40.446163241978304,-3.7075424194335938],
            "position": [40.45171508652634,-3.677501678466797],
            "password": "secret",
            "name": "c1",
            "fleet_type": "customer"
        }
    ]
```

Listing 5.2: Exemple de fitxer de configuració

A banda, també s'han de definir valors com el màxim temps de simulació, els fitxers d'estratègia de cada agent, el nom del host, el port XMPP, el port HTTP o la direcció IP d'HTTP.

Per a generar els fitxers de configuració que s'han utilitzat per a les proves, s'ha fet ús del generador de càrrega descrit a la secció 4.1.4. La idea principal era aconseguir distints resultats significatius per a poder contrastar les estratègies propostes.

En total s'han generat set fitxers on tots els agents apareixen al principi, però amb distintes quantitats de clients. S'han creat fitxers de 10, 15, 20, 25, 30, 50 i 75 clients. Cal recalcar que per a que els resultats foren més robusts i fàcils de comparar, l'aparició dels clients és incremental als fitxers, és a dir, els primers 10 clients del fitxer de 15 clients són els mateixos que els del fitxer de 10, i aquestos 15 seran els primers en aparèixer a la simulació amb 20 clients. D'altra banda, també s'han generat fitxers amb 30, 50 i 75 clients on van apareixent de forma gradual o dinàmica, per a veure possibles diferències amb l'altre tipus de configuració, on tots els clients estan des del principi.

A banda, tots els fitxers generen els clients amb una distribució aleatòria, és a dir, no s'ha seguit ninguna distribució informada prèviament. Els clients apareixen de forma incremental, però sense una raó o criteri respecte a la ubicació concreta.

CAPÍTOL 6
Experimentació

En el present capítol es presentaran els resultats obtinguts a les proves sobre la nova versió del simulador SimFleet.

Es començarà explicant la configuració de les proves realitzades, per a tindre un context abans de veure els resultats. Seguidament es mostraran els mencionats resultats, en diferents subseccions per a separar el tipus de proves, que es poden dividir en proves estàtiques i proves dinàmiques.

Al llarg del capítol es mostren diverses taules i gràfiques que ajuden a entendre els resultats, que s'acaben explicant i raonant a l'última secció, on es desenvolupa una breu discussió.

6.1 Configuració dels experiments

A continuació es mostraran i explicaran els resultats dels experiments proposats per a comprovar les prestacions de les quatre estratègies desenvolupades. Per a cada cas, s'ha realitzat una serie de proves canviant els valors dels paràmetres de capacitat i quantitat de clients.

S'ha decidit no utilitzar ninguna estació de recàrrega, ja que és un factor que no ens afecta al nostre objectiu, i per tant sols afegiria una complexitat innecessària. Tampoc s'ha fet ús de més d'un transport, ja que la idea es comprovar les prestacions per vehicle. Si férem ús de diversos vehicles, tan sols es reduiria el temps de la simulació, de forma quasi inversament proporcional a la quantitat de vehicles que s'afigen.

D'altra banda, la velocitat que s'utilitza per al vehicle és de 8000 kilòmetres per hora, el qual és un valor completament irreal però que ajuda a alleugerar el temps de realització dels experiments. Per tant, tenint en compte que a la ciutat es condueix de mitjana a 40 kilòmetres per hora, els valors que obtenim als resultats seran aproximadament unes 200 vegades més menuts que el que podrien ser a la realitat.

S'ha fet una serie de proves per a cada una de les estratègies amb capacitats de tres, cinc i set clients com a màxim al vehicle. Per a cada capacitat s'ha provat a executar una simulació de 10, 15, 20, 25, 30, 50 i 75 clients, els quals apareixen de forma estàtica (quan apareixen tots els clients al principi). A banda, s'han fet

proves amb aparició dinàmica dels clients (van apareixent en intervals de temps) amb capacitat de cinc passatgers amb 30, 50 i 75 clients.

Les mesures que s'utilitzaran per a comparar els diversos resultats seran les següents:

1. Temps d'espera mitjà per client: és la suma del temps que espera cada client a que el transport els arreplegue, dividit per la quantitat de clients.
2. Temps mitjà total per client: és la suma del temps que espera cada client per a ser arreplegat i deixat al seu destí, dividit per la quantitat de clients.
3. Distància mitjana: distància mitjana recorreguda pels vehicles de la simulació. Al nostre cas sols haurà un vehicle, per tant, el valor no serà ninguna mitjana, serà directament la distància que ha recorregut.
4. Temps de simulació: temps que passa des que s'inicia la simulació fins que es deixa l'últim client al seu destí.

És clar que la millor estratègia serà la que reduïsca en major quantitat aquestes quatre mesures.

6.2 Resultats

6.2.1. Proves estàtiques

Les proves estàtiques s'anomenen així perquè l'aparició de tots els client es dona al principi de la simulació. En aquest moment tots demanen servei al transport a la vegada i aquest comença a realitzar els viatges.

Capacitat de tres passatgers

Per a la menor de les capacitats, els resultats en cada una de les mètriques són pareguts en quan a la millor estratègia.

Es pot veure clarament com els temps d'espera (Taula 6.1) com temps total (Taula 6.2) dels clients són millors amb l'estratègia de proximitat, excepte a la prova de 10 clients, on el millor rendiment és de la DB. Açò canvia amb 15 clients, on la pitjor prestació és la de TSP. Aquest empitjorament de TSP s'explica per l'alta complexitat de l'algoritme, la qual creix amb el tamany de la llista de coordenades. En segon i tercer lloc a les proves amb més de 10 clients tenim DB i FIFO, respectivament. Amb 75 clients la millor és DB seguida de FIFO.

Els resultats de distància recorreguda (Taula 6.3) són lleugerament millors per a 10 clients amb TSP que amb proximitat, però amb més clients és aquesta segona la que millors resultats dona. A partir dels 20 clients és seguida per DB, i finalment FIFO. Per a 75 clients la millor és DB.

Finalment, el temps de simulació (Taula 6.4) mínim és obtingut, en totes les proves, per l'estratègia de proximitat, seguida de TSP en els casos de 10 i 15 clients. Es repeteixen la segona i tercera millors prestacions per a la resta de casos

per a DB i FIFO, respectivament. Aquestes dues són la millor i segona millor en el cas de 75 clients.

Cal recalcar que per a l'estratègia TSP no s'ha pogut realitzar experiments de més de 15 clients, ja que la complexitat de l'algoritme, tot i fent ús d'una tècnica eficient com és la programació dinàmica, no permet realitzar l'execució en un temps raonable. El fet d'haver de generar una llista ordenada comprovant distintes possibilitats amb cada vegada més coordenades genera una gran complexitat que no es pot assumir amb l'equip amb que es realitzen les proves. L'agent de transport necessita obtindre la tasca en pocs segons i, de no ser així, no pot funcionar correctament, per això s'han descartat eixos resultats. Sols es mostren els resultats per als fitxers de 10 i 15 clients ja que són les úniques configuracions que permeten l'execució de la simulació sense problemes temporals. De forma similar tampoc s'ha aconseguit realitzar la simulació amb 75 clients amb l'estratègia de proximitat.

Els resultats es poden veure també en forma de gràfiques a la figura 6.1, on es veu fins a les proves amb 50 clients com l'estratègia que millors resultats dona és la de proximitat. No es mostren les barres per a les proves amb 75 clients ja que tan sols hi havia resultats per a TSP i DB, per tant no aporta massa informació addicional.

Nº clients	FIFO	DB	Proximitat	TSP
10	36.85	31.11	15.22	**14.65**
15	55.93	47.13	**16.23**	39.08
20	70.62	53.50	**18.33**	-
25	88.94	82.44	**24.16**	-
30	107.31	87.72	**27.27**	-
50	171.85	144.55	**33.92**	-
75	265.59	**232.61**	-	-

Taula 6.1: Temps mitjà d'espera per client en segons amb tres passatgers de capacitat

Nº clients	FIFO	DB	Proximitat	TSP
10	40.81	35.70	30.16	**27.91**
15	59.78	51.38	**38.98**	62.38
20	74.38	57.48	**39.17**	-
25	92.75	86.62	**44.67**	-
30	111.17	92.21	**49.83**	-
50	175.71	149.06	**52.15**	-
75	269.51	**237.12**	-	-

Taula 6.2: Temps mitjà total per client en segons amb tres passatgers de capacitat

Capacitat de cinc passatgers

Per a la capacitat de cinc passatgers, els resultats també són bastant consistents i pareguts als obtinguts amb capacitat de tres clients.

Nº clients	FIFO	DB	Proximitat	TSP
10	121669.00	115884.00	66635.30	**66243.60**
15	183628.00	173885.00	**80944.50**	82052.30
20	231362.00	199276.00	**97368.50**	-
25	293180.00	286327.00	**111528.00**	-
30	350604.00	323398.00	**145924.00**	-
50	576466.00	528399.00	**291536.60**	-
75	875708.00	**837907.00**	-	-

Taula 6.3: Distància recorreguda en metres amb tres passatgers de capacitat

Nº clients	FIFO	DB	Proximitat	TSP
10	74.58	70.90	**42.61**	43.77
15	110.17	104.55	**51.67**	79.15
20	142.62	121.85	**63.24**	-
25	176.75	173.27	**74.38**	-
30	215.48	194.08	**93.48**	-
50	349.39	313.47	**182.39**	-
75	533.36	**495.19**	-	-

Taula 6.4: Temps de la simulació en segons amb tres passatgers de capacitat

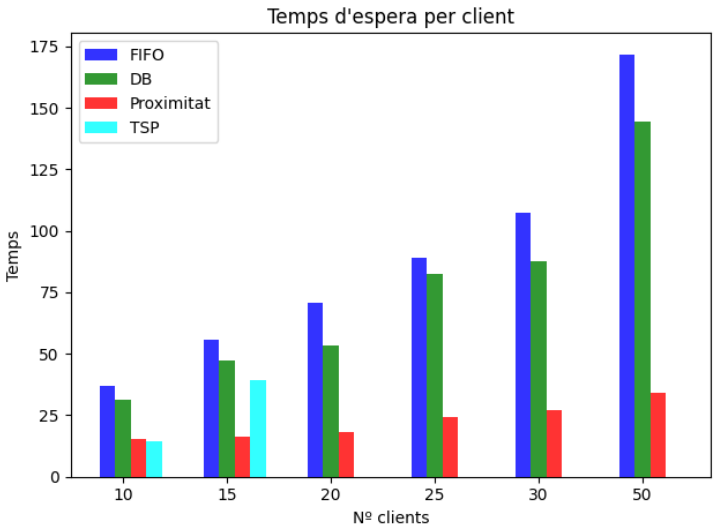
Es veu que el temps d'espera (Taula 6.5) en aquest cas és dominat per l'estratègia de proximitat, fins als 50 clients, i el temps total (Taula 6.6) dels clients és millor quan s'utilitza també l'estratègia de proximitat, excepte a la prova de 10 clients, on el millor rendiment és de la TSP. Amb 15 clients TSP empitjora fins al major temps. En segon i tercer lloc a les proves amb més de 10 clients tenim de nou DB i FIFO, respectivament. Per a 75 clients, com que no tenim dades per a proximitat, guanya DB en tots els casos.

Els resultats de distància recorreguda (Taula 6.7) són lleugerament millors per a 10 clients amb TSP que amb proximitat, però amb 15 clients és la de proximitat la millor, igual que amb la resta de capacitats. A partir dels 20 clients és seguida per DB, i finalment FIFO. Amb 75 clients la millor és DB.

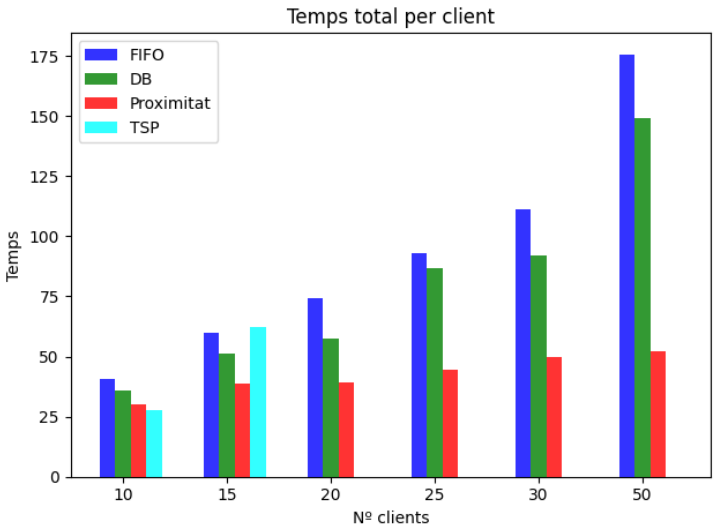
Per últim, el temps de simulació (Taula 6.8) mínim és obtingut, en totes les proves, per l'estratègia de proximitat, seguida un altra vegada de TSP en els casos de 10 i 15 clients. Es repeteixen la segona i tercera millors prestacions per a la resta de casos per a DB i FIFO, respectivament, les quals són primera i segona al cas de 75 clients.

De la mateixa manera que amb capacitat de tres passatgers, per a l'estratègia TSP no hi ha resultats de més de 15 clients, ja que no permet finalitzar l'execució degut a la seua complexitat computacional. Sols es mostren els resultats per als fitxers de 10 i 15 clients ja que són les úniques configuracions que permeten l'execució de la simulació sense problemes temporals. Tampoc s'ha aconseguit realitzar la simulació amb 75 clients amb l'estratègia de proximitat.

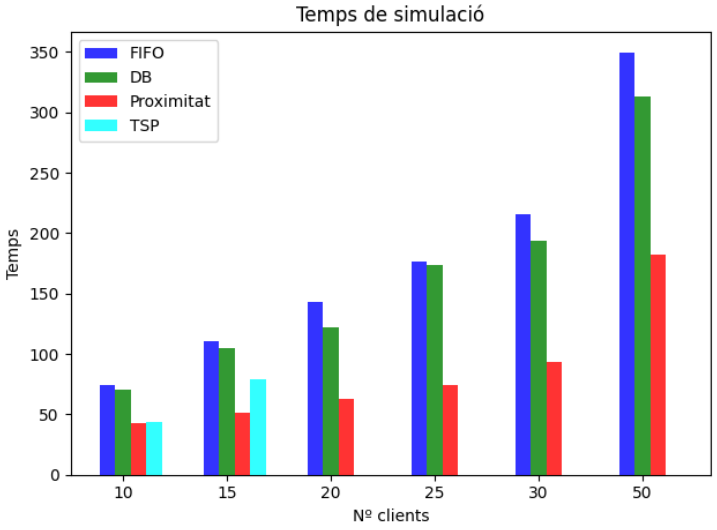
Els resultats estan representats també en gràfiques a la figura 6.2, on es veu també que l'estratègia que millors resultats dona és la de proximitat. No es mos-



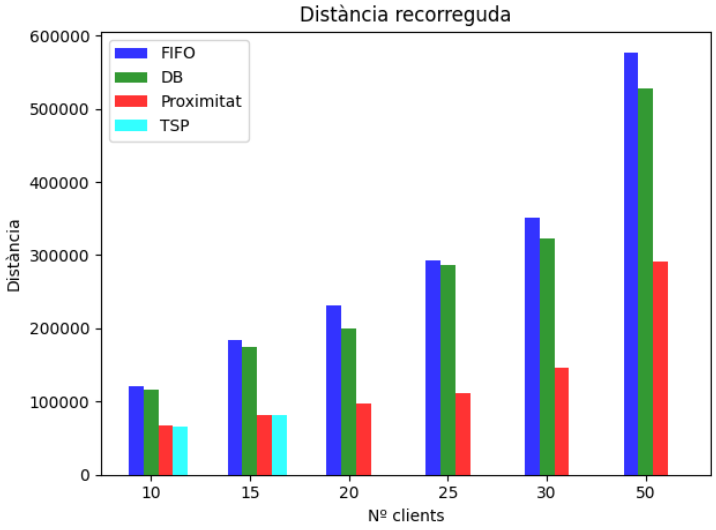
(a) Temps d'espera per client.



(b) Temps total per client.



(c) Temps de la simulació.



(d) Distància recorreguda.

Figura 6.1: Estadístiques per a capacitat de tres passatgers.

tren els valors amb 75 clients ja que tan sols s'han pogut executar la TSP i la DB amb eixa quantitat.

Nº clients	FIFO	DB	Proximitat	TSP
10	36.66	31.08	**14.38**	15.17
15	59.36	53.64	**16.29**	42.29
20	70.20	57.38	**18.40**	-
25	89.12	82.92	**23.93**	-
30	107.58	91.19	**28.43**	-
50	170.65	151.15	**40.77**	-
75	267.83	**221.51**	-	-

Taula 6.5: Temps mitjà d'espera per client en segons amb cinc passatgers de capacitat

Nº clients	FIFO	DB	Proximitat	TSP
10	40.65	35.79	28.88	**28.51**
15	63.45	58.24	**39.02**	66.21
20	73.86	61.51	**39.45**	-
25	92.92	87.12	**44.16**	-
30	111.39	95.86	**51.55**	-
50	174.50	155.71	**71.47**	-
75	271.80	**226.49**	-	-

Taula 6.6: Temps mitjà total per client en segons en segons amb cinc passatgers de capacitat

Nº clients	FIFO	DB	Proximitat	TSP
10	121669.00	115884.00	66635.30	**66243.60**
15	173885.00	183628.00	**80944.50**	82052.30
20	231362.00	199276.00	**97368.50**	-
25	293180.00	286327.00	**111528.00**	-
30	350604.00	323398.00	**145924.00**	-
50	576466.00	539061.00	**182768.00**	-
75	875708.00	**819825.00**	-	-

Taula 6.7: Distància recorreguda en metres en segons amb cinc passatgers de capacitat

Nº clients	FIFO	DB	Proximitat	TSP
10	73.82	71.38	**40.88**	43.98
15	116.84	116.07	**51.45**	82.67
20	141.42	128.38	**63.33**	-
25	177.38	173.38	**73.19**	-
30	219.48	201.49	**95.35**	-
50	345.43	324.30	**123.75**	-
75	541.25	**485.39**	-	-

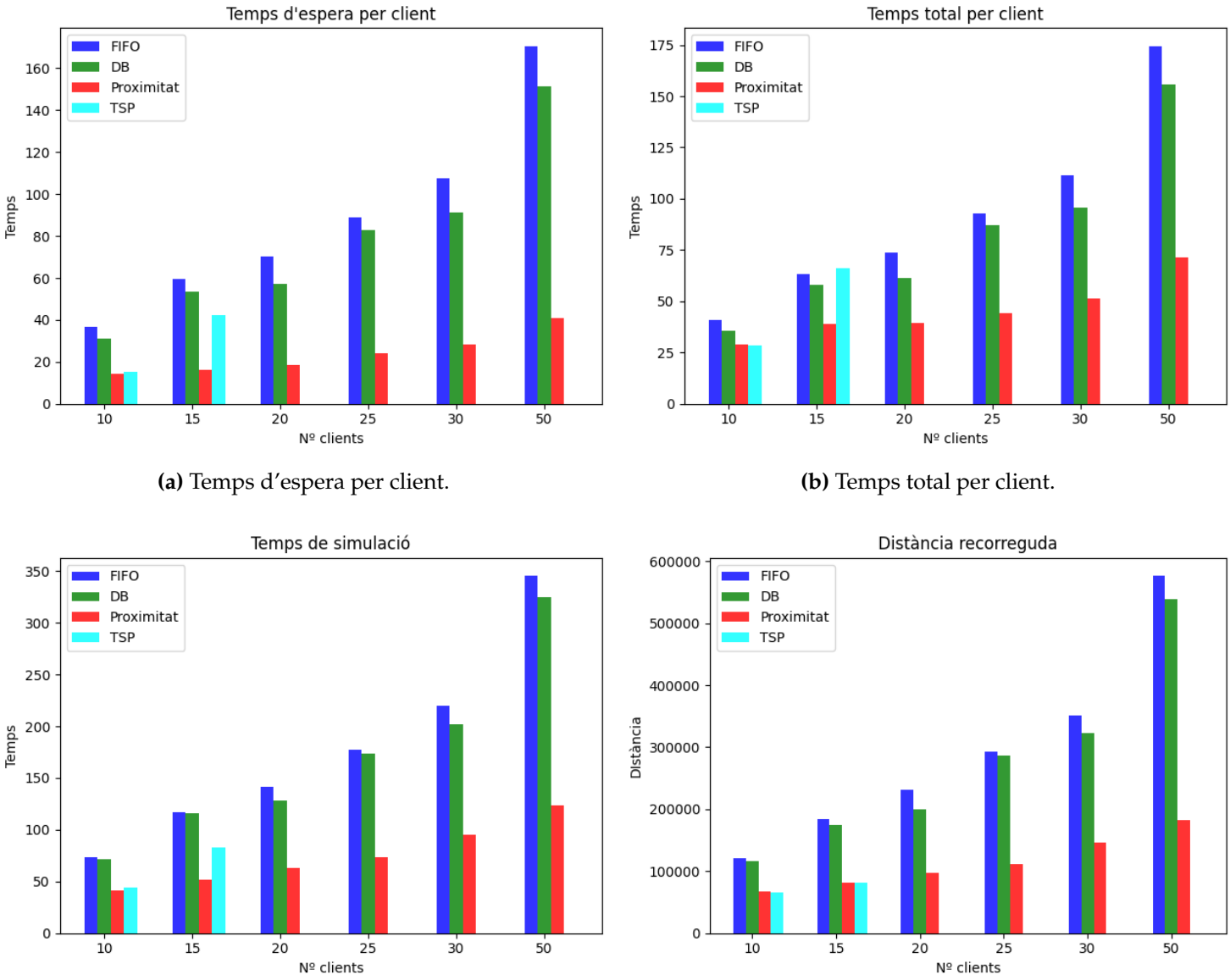
Taula 6.8: Temps de la simulació en segons amb cinc passatgers de capacitat

Capacitat de set passatgers

Finalment, a la capacitat de set passatgers els resultats també són molt pareguts als obtinguts amb menors capacitats.

Els temps d'espera (Taula 6.9) i temps total (Taula 6.10) dels clients són millors amb l'estratègia de proximitat, menys a la prova de 10 clients, on el millor rendiment és de la TSP, per poc. Amb 15 clients TSP empitjora fins al major temps degut a la seua complexitat computacional. En segon i tercer lloc a les proves amb més de 10 clients tenim de nou DB i FIFO, en eixe ordre, que són la primera i segona per a 75 clients.

Els resultats de distància recorreguda (Taula 6.11) són lleugerament millors per a 10 clients amb TSP que amb proximitat, però amb 15 clients és la de proximitat la millor, igual que amb la resta de capacitats. A partir dels 20 clients



(a) Temps d'espera per client.

(b) Temps total per client.

(c) Temps de la simulació.

(d) Distància recorreguda.

Figura 6.2: Estadístiques per a capacitat de cinc passatgers.

és seguida per DB, i finalment FIFO. En eixe mateix ordre són la millor i segona millor estratègies amb 75 clients.

Per acabar, el temps de simulació (Taula 6.12) mínim és obtingut en aquest cas per TSP en la prova de 10 clients, però amb 15 és superada per proximitat, la qual s'executa en el menor temps en la resta de casos. Es repeteixen de nou el segon i tercer millors temps per a la resta de casos per a DB i FIFO, respectivament, que a la vegada són primer i segon per a 75 clients.

Igual que amb amb les anteriors capacitats, ampliar-la a set passatgers no aconsegueix evitar que amb l'estratègia TSP no es puguen realitzar experiments de més de 15 clients, per la seua complexitat. Sols es mostren els resultats per als fitxers de 10 i 15 clients ja que són les úniques configuracions que permeten l'execució de la simulació sense problemes. En aquest cas tampoc s'ha aconseguit realitzar la simulació amb 75 clients amb l'estratègia de proximitat.

A la figura 6.3 es poden comprovar els resultats en forma de gràfiques, on es comprova també que l'estratègia que millors resultats dona és la de proximitat. No es mostren els valors amb 75 clients ja que tan sols s'han pogut executar la TSP i la DB, igual que en els casos de menors capacitats.

Nº clients	FIFO	DB	Proximitat	TSP
10	37.42	30.98	15.22	**14.84**
15	57.48	49.28	**16.25**	42.74
20	71.62	53.79	**18.87**	-
25	92.15	82.22	**24.90**	-
30	108.65	88.56	**27.99**	-
50	169.97	157.82	**41.85**	-
75	257.17	**225.40**	-	-

Taula 6.9: Temps mitjà d'espera per client en segons amb set passatgers de capacitat

Nº clients	FIFO	DB	Proximitat	TSP
10	41.49	35.65	33.53	**28.83**
15	61.46	53.70	**39.62**	66.36
20	75.38	57.78	**41.08**	-
25	96.10	86.39	**45.52**	-
30	112.55	93.07	**51.17**	-
50	173.81	162.11	**72.91**	-
75	260.98	**230.00**	-	-

Taula 6.10: Temps mitjà total per client en segons amb set passatgers de capacitat

Nº clients	FIFO	DB	Proximitat	TSP
10	121669.00	115884.00	66635.30	**66243.60**
15	183628.00	173885.00	**80944.50**	82052.30
20	231362.00	199276.00	**97368.50**	-
25	293180.00	286327.00	**111528.00**	-
30	350604.00	323398.00	**145924.00**	-
50	576466.00	546603.00	**182768.00**	-
75	875708.00	**821144.00**	-	-

Taula 6.11: Distància recorreguda en metres amb set passatgers de capacitat

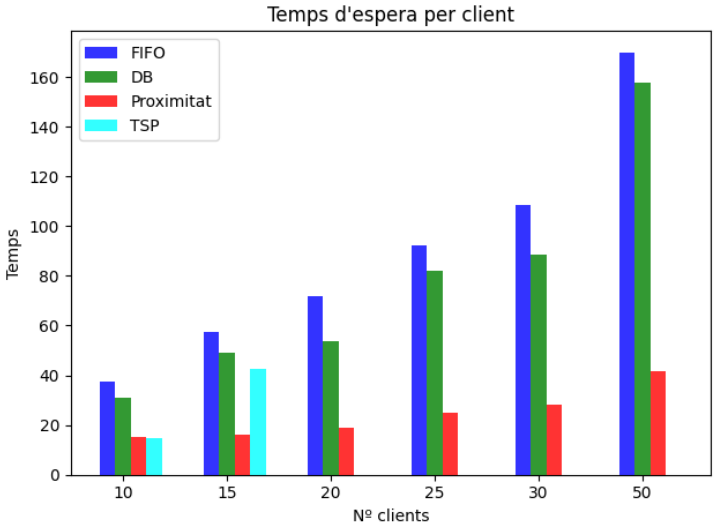
6.2.2. Proves dinàmiques

La idea de realitzar proves dinàmiques té com a propòsit fer unes simulacions més complexes però realistes a la vegada, ja que aquest funcionament s'assembla més a la realitat que el d'una execució on els clients apareixen tots al principi.

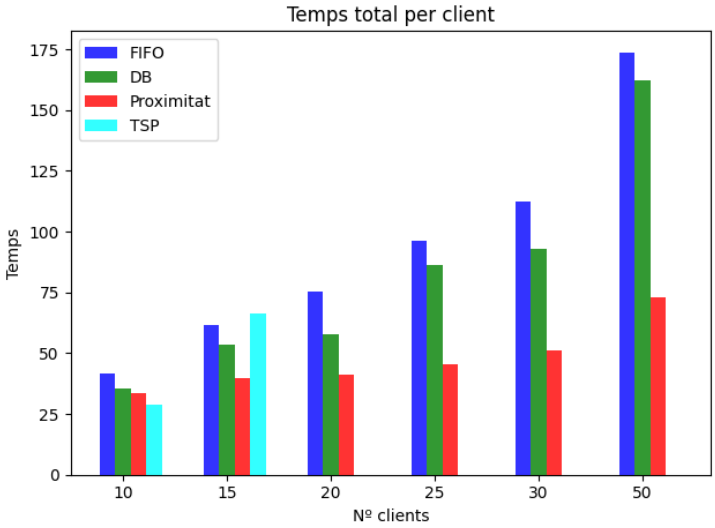
Una prova dinàmica és una prova en què els clients apareixen temporalment, és a dir, cada cert temps apareixen nous clients a banda dels que ja han sol·licitat servei i han sigut atesos o estan en espera.

Nº clients	FIFO	DB	Proximitat	TSP
10	75.72	71.14	50.32	**46.32**
15	113.31	109.23	**52.70**	82.42
20	145.44	122.23	**66.23**	-
25	183.54	172.44	**74.93**	-
30	217.21	194.85	**95.34**	-
50	345.96	331.35	**126.77**	-
75	520.03	**485.74**	-	-

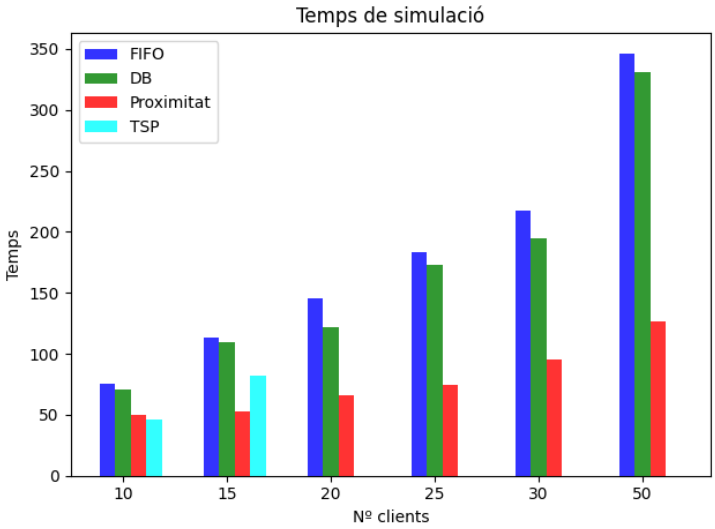
Taula 6.12: Temps de la simulació en segons amb set passatgers de capacitat



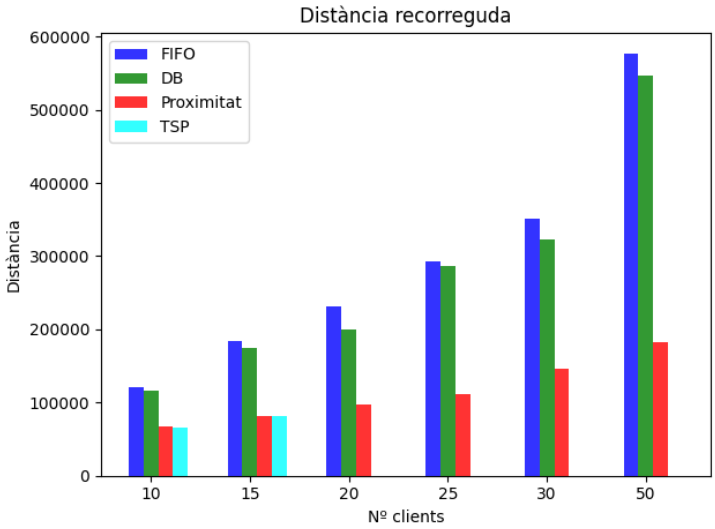
(a) Temps d'espera per client.



(b) Temps total per client.



(c) Temps de la simulació.



(d) Distància recorreguda.

Figura 6.3: Estadístiques per a capacitat de set passatgers.

Per a aquest cas, al veure que la capacitat no afecta significativament a les mesures de referència, sols s'han realitzat proves amb una capacitat, triant la de cinc passatgers per ser la mitjana de les capacitats provades anteriorment.

S'han generat fitxers de proves per a 30, 50 i 75 clients directament. La raó per la qual no s'han provat simulacions amb menys clients és per a poder veure l'efecte de l'aparició dinàmica dels clients, ja que amb pocs clients no donaria temps a poder apreciar-ho del tot.

En quan a les mesures de referència, podem observar a la Taula 6.13 com el millor temps mitjà per client és obtingut per TSP amb 30 clients, i per proximitat a la resta. La Taula 6.14 mostra també el domini clar de TSP amb 30 clients i un empitjorament en aquest cas de l'estratègia de proximitat, la qual remunta a la millor prestació als casos amb més clients. La distància recorreguda, que es pot veure a la Taula 6.16, mostra igualment que amb 30 clients s'obté millor resultat amb TSP i amb 50 i 75, amb proximitat. Finalment, a la Taula 6.15 s'aprecia que el menor temps de simulació és amb TSP per a 30 clients, i als altres dos casos, de nou, amb l'estratègia de proximitat.

Els resultats obtinguts han seguit el mateix patró que amb les proves estàtiques. La estratègia TSP sols s'ha pogut provar amb 30 clients, el qual és una quantitat major que la màxima a la que arribava anteriorment, 15. Açò és degut a que al aparèixer els clients dinàmicament, l'algoritme tarda més en sobrecarregar-se. Tot i això, les proves amb 50 i 75 clients ja no les suporta.

La millor estratègia als experiments amb 50 i 75 persones és la de proximitat, amb una diferència sobre DB i FIFO que augmenta amb el nombre de clients. El pitjor resultat per a totes les capacitats és, com a tots els experiments realitzats i com és d'esperar, amb l'estratègia FIFO. A la figura 6.4 es poden veure més gràficament aquests conclusions.

Els temps mitjà i totals d'espera milloren bastant respecte a les proves estàtiques. L'explicació a açò és que al aparèixer més tard la majoria de clients, el temps que transcorre mentre esperen i fins que arriben a la seua destinació és menor que si ja estan esperant des d'un principi.

La distància recorreguda no millora si comparem amb les mateixes estratègies i capacitat, degut a que el client més pròxim s'elegeix entre tots els que queden a les proves estàtiques, per la qual cosa es recorre menys distància. En les proves dinàmiques s'elegeix cada vegada a un client dels que ja han aparegut, el qual no ha de ser el més proper tenint en compte també els que queden per aparèixer. El mateix raonament s'aplica per al temps total de simulació, ja que va directament relacionat a la distància que recorre el transport.

Nº clients	FIFO	DB	Proximitat	TSP
30	66.19	59.70	47.63	**20.67**
50	123.16	92.73	**33.06**	-
75	188.39	126.24	**36.32**	-

Taula 6.13: Temps mitjà d'espera per client en segons amb cinc passatgers de capacitat(aparició dinàmica dels clients)

Nº clients	FIFO	DB	Proximitat	TSP
30	70.00	64.74	74.67	**33.33**
50	126.98	97.73	**50.95**	-
75	192.19	131.59	**56.81**	-

Taula 6.14: Temps mitjà total per client en segons amb cinc passatgers de capacitat(aparició dinàmica dels clients)

Nº clients	FIFO	DB	Proximitat	TSP
30	347900.00	331409.00	201905.00	**201140.00**
50	589010.96	514499.35	**305802.77**	-
75	920436.03	762660.74	**402962.00**	-

Taula 6.15: Distància recorreguda en metres amb cinc passatgers de capacitat(aparició dinàmica dels clients)

Nº clients	FIFO	DB	Proximitat	TSP
30	211.46	197.66	179.45	**128.37**
50	349.88	308.88	**192.64**	-
75	537.45	451.77	**251.45**	-

Taula 6.16: Temps de la simulació en segons amb cinc passatgers de capacitat(aparició dinàmica dels clients)

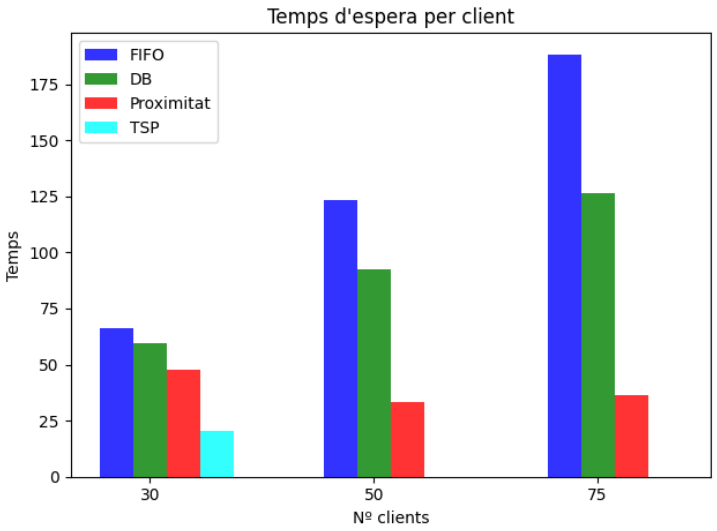
6.3 Discussió

Una vegada vistos tots els resultats tant en taules com en gràfiques, podem arribar a diverses conclusions.

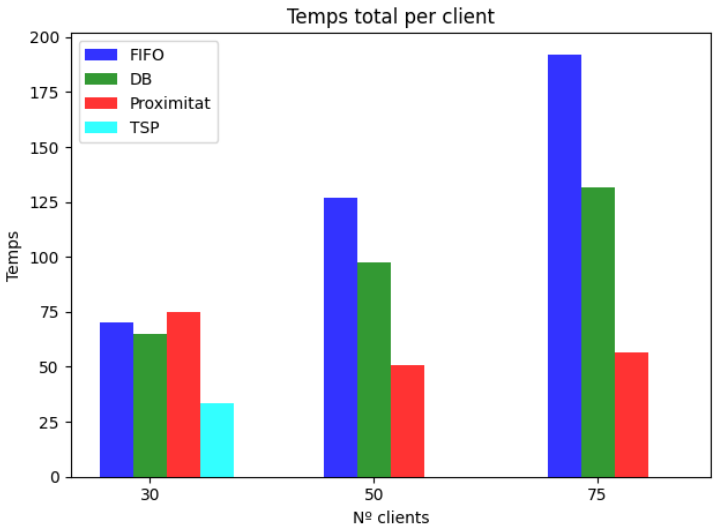
Primerament, es pot veure com l'ús de les tres capacitats no genera quasi diferències entre elles en quant a les mètriques estudiades. En un primer moment es podia pensar que seria un paràmetre més rellevant, però s'ha acabat veient que no és així.

Respecte a les estratègies, com era de suposar, la pitjor ha sigut la FIFO, ja que és la més bàsica. El seu comportament és el menys eficient ja que no té en compte ninguna distància en el moment d'elegir la ubicació a la qual anar. Tot i això, és útil tindre els resultats amb aquesta tàctica, ja que així es pot valorar la millora de la resta respecte a aquesta.

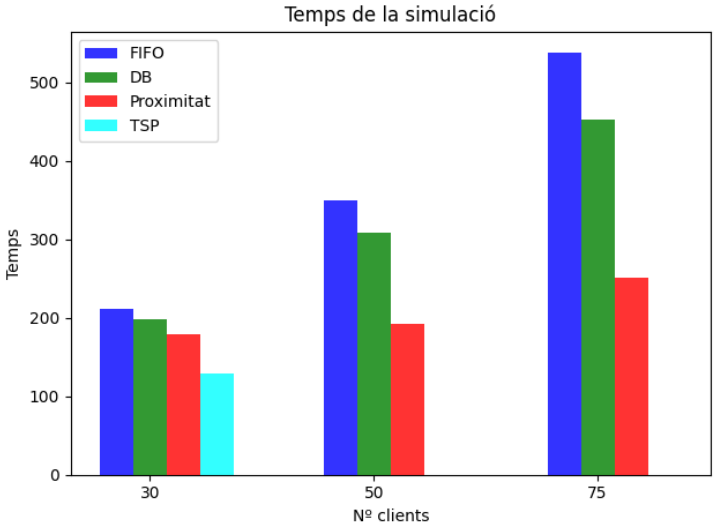
La estratègia de distància balancejada ha donat resultats pitjors dels esperats. Es podia pensar que anaven a ser similars als de proximitat, o almenys a millorar en els temps totals de cada client, ja que tracta de prioritzar l'atenció als passatgers que ja són al vehicle. Però ninguna de les dos previsions s'ha complit. Probablement, aquesta mala prestació és deguda al sistema de ponderació per a la prioritat segons el temps al vehicle, explicat al capítol anterior. És complicat poder encertar els valors exactes per a millorar el temps total de mitjana per client sense que les prestacions generals de la simulació es vegen negativament afectades.



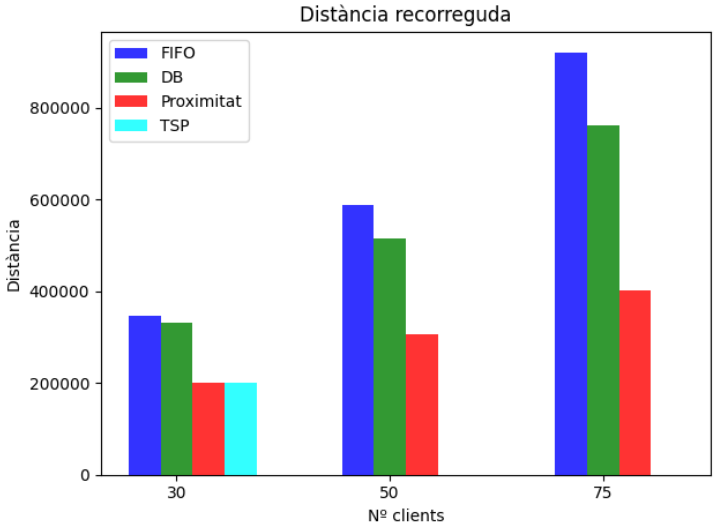
(a) Temps d'espera per client.



(b) Temps total per client.



(c) Temps de la simulació.



(d) Distància recorreguda.

Figura 6.4: Estadístiques per a capacitat de cinc passatgers (aparició dinàmica dels clients).

En el cas de l'estratègia de proximitat, s'ha vist que, de forma general, és la millor. És l'estratègia que més redueix les mètriques estudiades a la majoria de proves, ja que TSP sols es computable als experiments més menuts. Aquesta estratègia no compta amb una gran complexitat, més que la de ordenar per proximitat una serie de coordenades, i el resultat és més que satisfactori.

La última estratègia, TSP, dona els millors resultats amb pocs clients, la qual cosa ens du a pensar que podria ser la més eficient en la resta de casos si la seua execució fora menys complexa. Tot i això, en els casos que destaca, la seua diferència amb la tàctica de proximitat no és massa gran, però suficient per a comprovar que en aquestos casos buscar un ordre òptim per a visitar una serie de coordenades és un poc més eficient que anar a per la coordenada més propera en cada moment.

Finalment, si es comparen els resultats de les proves estàtiques amb els de les proves dinàmiques, com es menciona a la secció anterior, milloren els temps d'espera i total per client de mitjana. La raó és que al aparèixer dinàmicament els clients, no esperen tant per a arribar a la seua destinació com si apareixen tots al principi directament. Aquesta diferència es nota més amb l'augment de client per simulació, per la mateixa raó. També s'han obtingut millors resultats que en les proves estàtiques amb 30, 50 i 75 clients gràcies a que, al reduir-se la complexitat de les proves per tindre els clients més espaiats temporalment, s'han pogut utilitzar les estratègies més eficients a proves que de forma estàtica no es podia. És el cas de TSP, que s'ha pogut provar amb la configuració de 30 clients d'aparició dinàmica, però no amb més de 15 clients d'aparició estàtica. L'estratègia de proximitat s'ha pogut provar amb 75 clients, el qual no era possible amb l'experiment estàtic.

Pel contrari, les proves d'aparició dinàmica resulten pitjors en quant a temps de simulació i distància recorreguda, ja que no és possible optimitzar els recorreguts si no es té una visió global de totes les ubicacions, el qual si es té amb les proves d'aparició estàtica.

En resum, independentment de si els clients apareixen tots al principi o no, s'ha vist que la millor estratègia generalment és la de proximitat. No podem dir que és superada per la de TSP ja que no s'han pogut obtindre resultats més amb els fitxers menuts, però és molt probable que aquesta fora la millor estratègia en tots els casos. Això seria possible si l'algoritme TSP es poguera executar més ràpid, independentment de la quantitat de clients. L'estratègia de DB ha donat millors resultats que la FIFO però no per la diferència esperada, pel qual es pot concloure que balancejar la distància per donar prioritat a certs clients no millora els resultats en ninguna de les mètriques.

CAPÍTOL 7
Conclusions

En aquest capítol es repassen els objectius principals del projecte, per a determinar si s'han aconseguit i en quina mesura.

S'han avaluat satisfactòriament els simuladors de transport existents per acabar elegint SimFleet, el que més avantatges oferia, i s'han revisat alguns estudis previs similars per entendre els seus objectius i conclusions, que han servit d'ajuda.

El plantejament del sistema a modificar ha sigut una de les parts més complexes i importants, ja que el desenvolupament depèn d'un bon plantejament. S'ha aconseguit plantejar els canvis necessaris al codi de SimFleet per a canviar el tipus de vehicle a un bus de servei sota demanda o 'demand-responsive'.

L'objectiu de desenvolupar el codi per a modificar el funcionament del simulador ha sigut acomplit amb èxit. S'ha arribat a modificar el sistema per a que el transport funcione amb dos màquines d'estats paral·leles, amb el que s'ha pogut obtindre un sistema demand-responsive amb un vehicle amb capacitat de clients variable.

S'ha aconseguit també dissenyar un total de quatre estratègies per a determinar l'ordre en que es serveix als clients, per a provar posteriorment quina funciona millor i en quins casos.

La fase de validació s'ha aconseguit amb èxit, realitzant diverses proves canviant diversos paràmetres per a comprovar la robustesa del sistema. Quan s'ha detectat algun problema s'ha pogut resoldre sense complicacions.

En quant als resultats dels experiments, s'ha obtingut una gran quantitat d'estadístiques que han servit de gran utilitat per analitzar-les posteriorment. En aquest punt, no s'han aconseguit tots els resultats desitjats, ja que per a algunes estratègies i mides de fitxers no s'ha pogut completar la simulació. Tot i això, s'ha aconseguit recopilar una gran quantitat de resultats, suficients per a la realització d'un anàlisi de qualitat. Aquests resultats s'han representat en forma de taules i gràfiques per a una millor comprensió, el qual era un dels sub-objectius principals també.

Finalment, l'extracció de conclusions s'ha realitzat satisfactòriament gràcies als resultats prèviament obtinguts de les proves. Respecte a la qüestió principal de quina és la millor estratègia, els millors resultats han sigut obtinguts per l'estratègia TSP quan s'ha pogut executar sense problemes. La seua eficiència recau

en que funciona obtenint la propera localització a la que viatjar amb vista als següents punts de la ruta, minimitzant així el recorregut dels viatges de la jornada. Si tenim en compte la majoria de resultats, aleshores la millor estratègia ha sigut la de proximitat, degut a que es centra en la tasca localitzada més prop al transport en eixe moment, una idea simple i que a la pràctica és efectiva.

A nivell general, es pot considerar que el projecte s'ha realitzat amb èxit al acomplir l'objectiu principal de desenvolupar simulacions de transport adaptat a la demanda i comprovar les prestacions de distintes estratègies ideades. El codi modificat del simulador es pot veure a a la branca 'demand-responsive' del repositori de GitHub de SimFleet [1].

Personalment, aquest projecte m'ha servit per a millorar diversos conceptes de la informàtica i la programació, i en concret del llenguatge Python. He après i ampliat algunes nocions de la Intel·ligència Artificial, el qual és el meu camp de major interès dins del vist al grau d'Enginyeria Informàtica. Ha sigut un treball més complex que els realitzats fins a l'actualitat, que m'ha servit per a saber enfrontar-me a reptes majors als que estava acostumat i a mantindre una constància i dedicació que estic segur que em serà de gran utilitat per a futurs projectes o treballs.

[1] https://github.com/jaumejordan/simfleet/tree/feature/demand-responsive

CAPÍTOL 8
Relació del treball desenvolupat amb els estudis cursats

Respecte a algunes de les assignatures cursades durant els estudis de grau, es pot extraure certa relació amb aquest treball. Per ordre de major a menor relació, aquestes són les assignatures que més tenen a veure amb el treball:

- AIN (Agents Intel·ligents): aquesta és probablement l'assignatura amb més relació amb el projecte realitzat. S'introdueix el concepte d'agents intel·ligents, amb les seues característiques i propietats principals. A banda, es realitzen pràctiques en un simulador d'agents, PyGomas [1] . Aquest programa funciona sobre Python i també es basa en SPADE.

  Ha sigut un gran avanç conèixer d'avant-mà el sistema en el que finalment s'ha treballat, ja que ha facilitat la familiarització amb l'entorn de treball.

- SIN (Sistemes Intel·ligents): és l'assignatura prèvia a Agent Intel·ligents, ja que introdueix el concepte de sistemes intel·ligents, el qual també engloba agents. S'estudien i realitzen pràctiques amb algoritmes de intel·ligència artificial, d'on cal destacar el disseny i resolució de problemes basats en estats. Aquest tipus de problemes es resolen amb el programa CLIPS [2], que s'utilitza per a desenvolupar sistemes basats en regles, el qual comparteix certa similitud amb les màquines d'estats.

- TAL(Teoria d'Autòmats i Llenguatges): l'assignatura de TAL introdueix el concepte d'autòmats finits, els quals es poden representar en llenguatges. Aquests autòmats es poden representar com a màquines d'estats, amb unes transicions definides entre aquestos estats.

  Aquesta és l'assignatura que més coneixement aporta a la carrera sobre les màquines d'estats, i ha sigut de gran ajuda entendre eixos conceptes per a dissenyar i elaborar el sistema de comportaments a SimFleet, el qual com hem vist, es basa en màquines d'estats.

- EDA (Estructura de Dades i Algoritmes): el curs d'estructura de dades i algoritmes és un dels més generals de la carrera, però que al mateix temps, més utilitat té. Adquirir el raonament i habilitat per a resoldre problemes

[1] https://github.com/javipalanca/pygomas
[2] https://www.clipsrules.net/

utilitzant algoritmes eficients és un tret bàsic per al camp de l'enginyeria informàtica.

En quant a la relació directa amb el treball, s'han utilitzat estructures de dades, com diccionaris, llistes o arrays, per a tractar la informació dins del programa. Els diccionaris de Python han sigut molt útils a l'hora de guardar les dades dels clients que accepta el transport, ja que per cada client (clau) es volen guardar valors com la ubicació o el destí (valor, que a la seua vegada és un altre diccionari amb les seues claus i valors).

D'altra banda, es tracten una serie d'algoritmes per a minimitzar distàncies recorregudes, com Dijkstra [3], que ha sigut d'ajuda per a acabar elegint l'algoritme de TSP com a una de les estratègies.

- ALG (Àlgebra): igual que EDA, en Àlgebra es tracten alguns algoritmes bàsics per a la cerca o recorregut al llarg d'una sèrie de nodes. L'algoritme TSP també té a veure amb aquesta assignatura.

- ISW (Enginyeria del Software): d'aquesta assignatura s'ha pogut aplicar la metodologia de treball, ja que tot software ha de seguir un cicle, marcat per la planificació, el desenvolupament i proves, i la validació.

- IPC (Interfícies Persona Computador): de forma molt similar a l'anterior, ja que és la continuació, s'han pogut extraure el modelatge de maquetes i esbossos del sistema abans d'implementar-lo, una pràctica que a la llarga estalvia molt de temps.

En general, quasi totes les assignatures del Grau d'Enginyeria Informàtica han aportat coneixement necessari per a realitzar aquest treball. Des de les assignatures més bàsiques de programació (IIP, PRG, LTP) fins a d'altres més teòriques però que aporten conceptes de planificació i organització, com GPR. Per això es pot dir que totes les assignatures de la carrera han aportat en certa mesura per a poder realitzar aquest projecte.

[3] https://ca.wikipedia.org/wiki/Algorisme_de_Dijkstra

CAPÍTOL 9
Treballs futurs

En aquest últim capítol es mencionen algunes possibles millores o ampliacions al treball realitzat.

Per a un possible treball futur, es podrien provar altres estratègies diferents a les plantejades, ja que sols s'han testejat quatre d'una llarga llista de possibilitats. També seria ideal poder testejar l'estratègia de TSP amb majors dimensions de clients, ja que amb pocs clients ha resultat ser una de les estratègies que millors resultats genera. Per a poder realitzar-ho es necessitaria un processador suficientment potent per a realitzar els càlculs en pocs segons.

Tenint en compte aquesta possible millora de les prestacions de l'equip per a fer els experiments, seria ideal provar amb fitxers de més clients, amb 100 o més, per a extraure més conclusions. Amb aquestes quantitats de clients majors es podria comprovar com funciona el sistema amb l'aparició escalonada dels clients, és a dir, que no apareguen tots des d'un inici com s'ha fet a les proves estàtiques, que han sigut la majoria de les fetes al projecte.

Una altra ampliació seria fer proves amb més busos o vehicles, o de forma multi-modal, és a dir, amb una barreja de tipus de transport. Un exemple podria ser fer simulacions amb tres transports, un amb capacitat de tres passatgers, un altre amb capacitat de cinc i un altre de set. S'haurien de provar distintes distribucions per a veure quina s'adapta millor a la demanda.

D'altra banda, es pot plantejar fer simulacions amb mapes d'altres ciutats. La idea seria comparar els resultats obtinguts en distintes ciutats per a comprovar si són similars o cada estratègia funciona de forma diferent depenent del tipus de mapa.

Bibliografia

[1] Traveling Salesman Problem – Dynamic Programming Approach. https://www.baeldung.com/cs/tsp-dynamic-programming, 2022. Accedit el 27-06-2022.

[2] Sistemas multiagente. https://es.m.wikipedia.org/wiki/Sistema_multiagente, 2022. Accedit el 16-06-2022.

[3] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.

[4] R Bordini and Jomi F Hübner. An overview of jason. *Association for Logic Programming Newsletter*, 19(3), 2006.

[5] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. Jade: a fipa2000 compliant agent development environment. In *Proceedings of the fifth international conference on Autonomous agents*, pages 216–217, 2001.

[6] Lars Braubach, Alexander Pokahr, and Winfried Lamersdorf. Jadex: A short overview. In *Main Conference Net. ObjectDays*, volume 2004, pages 195–207. AgentExpo, 2004.

[7] PADE. https://pade.readthedocs.io/en/latest/, 2022. Accedit el 30-06-2022.

[8] Javier Palanca, Andrés Terrasa, Vicente Julian, and Carlos Carrascosa. Spade 3: Supporting the new generation of multi-agent systems. *IEEE Access*, 8:182537–182549, 2020.

[9] Michael Balmer, Marcel Rieser, Konrad Meister, David Charypar, Nicolas Lefebvre, and Kai Nagel. Matsim-t: Architecture and simulation times. In *Multi-agent systems for traffic and transportation engineering*, pages 57–78. IGI Global, 2009.

[10] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[11] AnyLogic. https://en.wikipedia.org/wiki/AnyLogic/, 2022. Accedit el 16-06-2022.

[12] Behnam Torabi, Mohammad Al-Zinati, and Rym Z Wenkstern. Matisse 3.0: A large-scale multi-agent simulation system for intelligent transportation systems. In *International conference on practical applications of agents and multi-agent systems*, pages 357–360. Springer, 2018.

[13] Vissim. https://www.ptvgroup.com/es/soluciones/productos/ptv-vissim-nuevo/, 2022. Accedit el 16-06-2022.

[14] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[15] Javier Palanca, Andrés Terrasa, Carlos Carrascosa, and Vicente Julián. Simfleet: a new transport fleet simulator based on mas. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 257–264. Springer, 2019.

[16] Pei-Pei Zhou. *Modelado de sistemas de free-floating carsharing mediante MATSim*. PhD thesis, Universitat Politècnica de València, 2021.

[17] Pasqual Martí, Jaume Jordán, Fernando De la Prieta, Holger Billhardt, and Vicente Julian. Demand-responsive shared transportation: A self-interested proposal. *Electronics*, 11(1):78, 2021.

[18] Sebastian Hörl. A matsim scenario for autonomous vehicles in la défense and île-de-france. *Arbeitsberichte Verkehrs-Und Raumplanung*, 1239, 2017.

[19] Francesco Ciari, Benno Bock, and Michael Balmer. Modeling station-based and free-floating carsharing demand: Test case study for berlin. *Transportation Research Record*, 2416(1):37–47, 2014.

[20] Pasqual Martí Gimeno. Improving urban mobility with game theory techniques. 2020.

[21] Pasqual Martí, Jaume Jordán, Javier Palanca, and Vicente Julian. Free-floating carsharing in simfleet. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 221–232. Springer, 2020.

[22] Pasqual Martí, Jaume Jordán, Javier Palanca, and Vicente Julian. Load generators for automatic simulation of urban fleets. In *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pages 394–405. Springer, 2020.

[23] Gerhard Reinelt. *The traveling salesman: computational solutions for TSP applications*, volume 840. Springer, 2003.

APÈNDIX A
Glossari

A.1 Abreviatures, acrònims i sigles

- **SMA**(Sistema Multi-Agent): sistema compost per múltiples agents intel·ligents que interactuen entre ells. Els sistemes multiagent poden ser utilitzats per resoldre problemes que són difícils o impossibles de resoldre per a un agent individual.

- **XML**(eXtensible Markup Language): es tracta d'un metallenguatge (un llenguatge que es fa servir per dir alguna cosa sobre un altre) extensible d'etiquetes que va ser desenvolupat pel Word Wide Web Consortium (W3C), una societat mercantil internacional que elabora recomanacions per a la World Wide Web.

- **FIPA**(Foundation for Intelligent Physical Agents)[1]: organització d'estàndards per a agents i sistemes multiagent oficialment acceptada per la IEEE [2] el vuit de juny de l'any 2015. L'objectiu d'aquesta organització és dedicarse a promoure la indústria dels agents intel·ligents a base de desenvolupar especificacions que donin suport a la interoperabilitat entre agents i aplicacions basades en agents

- **JSON**(JavaScript Object Notation)[3]: estàndard obert basat en text dissenyat per a intercanvi de dades llegible per humans. Deriva del llenguatge script JavaScript, per a representar estructures de dades simples i llistes associatives, anomenades objectes. Malgrat la seua relació amb el JavaScript, té implementacions per a gran part dels llenguatges de programació.

- **BDI**(Belief–Desire–Intention): el model de programari creença-desig-intenció és un model de programari desenvolupat per programar agents intel·ligents. Caracteritzat superficialment per la implementació de les creences, els desitjos i les intencions d'un agent, en realitat utilitza aquests conceptes per resoldre un problema particular en la programació d'agents.

[1] http://www.fipa.org/
[2] https://www.ieee.org/
[3] https://www.json.org/

- **OMS**(Open Street Map[4]): projecte col·laboratiu per crear mapes de contingut lliure usant dades obtingudes mitjançant dispositius GPS mòbils, ortofotografies i altres fonts de dades.

- **ITS**(Intelligent Transport Systems): el concepte de sistemes intel·ligents de transport és un conjunt de solucions tecnològiques de les telecomunicacions i la informàtica (coneguda com a telemàtica) dissenyades per millorar l'operació i la seguretat del transport terrestre, tant per a carreteres urbanes i rurals, com per a ferrocarrils.

- **GUI**(Graphical User Interface): interfície d'usuari gràfica que utilitza un conjunt d'imatges i objectes gràfics per representar la informació i accions disponibles a la interfície, en lloc d'una visualització purament textual per a un ordinador.

- **API**(Application Programming Interface): una interfície de programació d'aplicacions és una interfície que especifica com diferents components de programes informàtics haurien d'interaccionar. Dit d'una altra manera, és un conjunt d'indicacions, quant a funcions i procediments, ofert per una biblioteca informàtica o programoteca per ser utilitzat per un altre programa per interaccionar amb el programa en qüestió.

- **XMPP**(Extensible Messaging and Presence Protocol): protocol lliure de missatgeria instantània d'especificacions obertes basat en XML

- **FIFO**(First In First Out): Estratègia utilitzada al projecte per a obtindre el destí al que viatjar en una simulació de SimFleet,de la forma més simple. Es basa en el mètode del mateix nom, despatxant cada client tal com entra al transport, seguint el principi de "primer en entrar, primer en eixir".

- **DB**(Distància Balancejada): estratègia utilitzada al projecte per a obtindre el destí al que viatjar en una simulació de SimFleet, basant-se en el de menor distància de forma balancejada, amb un valor que pondera de 0 a 1 segons el temps que porta un passatger al transport.

A.2 Termes

- **Agent intel·ligent**: un agent intel·ligent és una entitat de programari que, basant-se en el seu propi coneixement, realitza un conjunt d'operacions destinades a satisfer les necessitats d'un usuari o d'un altre programa, bé per iniciativa pròpia o perquè algun dels anteriors li ho requereix

- **Host**: un amfitrió o amfitrió de xarxa és una computadora o un altre dispositiu connectat a una xarxa informàtica. Un servidor de xarxa pot oferir recursos, serveis i aplicacions d'informació als usuaris o altres nodes de la xarxa. Un amfitrió de xarxa és un node de xarxa que s'assigna una adreça de xarxa.

[4] https://www.openstreetmap.org/

- **Demand Responsive Transport**: forma de transport privat o quasi públic compartit per a grups que viatgen on els vehicles modifiquen les seues rutes a cada trajecte en funció de la demanda de transport particular sense utilitzar una ruta fixa o trajectes amb horaris. Aquests vehicles solen recollir i deixar passatgers en llocs segons les necessitats dels passatgers. i poden incloure taxis, autobusos o altres vehicles.

- **Multi-plataforma**: tipus d'aplicació/programa/programari que funciona en diversos sistemes operatius o dispositius, que sovint s'anomenen plataformes. Una plataforma significa un sistema operatiu com ara Windows, Mac OS, Android o iOS.

- **Multi-llenguatge**: que utilitza o integra diversos llenguatges de programació

- **GeoJSON**[5]: Format d'estàndard obert dissenyat per representar elements geogràfics senzills, juntament amb els seus atributs no espacials. Està basat en el format JSON.

- **Flota d'agents**: una flota és un conjunt d'agents, en el cas d'aquest projecte vehicles de transport, que es desplega amb una determinada quantitat en un mapa.

- **Autòmat finit**: model matemàtic d'un sistema compost per estats, transicions i accions. Un estat emmagatzema informació del passat. Una transició indica un canvi d'estat i es descriu per la condició que és necessària acomplir per activar la transició. Una acció és una descripció d'una activitat que es realitza en un moment donat. És sinònim de màquina d'estats.

[5] https://geojson.io/

APÈNDIX B
ODS

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**				X
ODS 10. **Reducción de las desigualdades.**				X
ODS 11. **Ciudades y comunidades sostenibles.**	X			
ODS 12. **Producción y consumo responsables.**			X	
ODS 13. **Acción por el clima.**			X	
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**				

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Actualment el desenvolupament de noves tecnologies no pot, o no ha de tindre una altra perspectiva que la que siga sostenible per al medi ambient. Encara que no siga directament, sempre caldria tindre en compte que el nostre producte es comprometa amb objectius que respecten el medi ambient o la societat en general. És de gran importància aconseguir un desenvolupament sostenible per poder seguir aprofitant el nostre entorn els propers anys.

Tot i que no de manera directa, ja que no era la idea principal, aquest treball en qüestió abasta diferents objectius per a un desenvolupament sostenible. El TFG es basa en simulacions de transport elèctric en què es busca obtindre proves experimentals per treure conclusions sobre els millors paràmetres o estratègies a seguir, prenent com a mesures de referència la distància total recorreguda pel vehicle per atendre totes les peticions de clients, el temps d'espera mitjà d'aquests clients, el temps de simulació...

Com que la idea és reduir la distància recorreguda mitjançant tècniques per trobar el camí més curt o el punt més proper, podem dir que el treball aporta el seu gra de sorra a l'ODS11, Çiutats i comunitats sostenibles", ja que reduir un trajecte en un vehicle per la ciutat suposa reduir la contaminació. Per la mateixa raó, la reducció de combustible utilitzat suposa una acció pel clima (ODTS 13,"Acció pel clima"); encara que no siga un projecte enfocat a aconseguir un canvi o una acció per a la millora del clima, la disminució de les emissions aporta una xicoteta ajuda per a contaminar menys. També suposa un consum responsable d'aquest combustible (ODTS 12, "Producció i consum responsables"), ja que el sistema està dissenyat per transportar més d'un client per viatge, per la qual cosa es rendibilitza molt cada desplaçament.

Podem dir que aquest treball fomenta el desenvolupament sostenible, ja que redueix el trànsit de la ciutat en ser més adaptable que un autobús però més sostenible que un taxi.