FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Sophisticated Agent Modeling: Language Design and Implementation for Large-Scale Simulation

José Eduardo Henriques

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Pedro Diniz

Co-Supervisor: António Costa

July 22, 2024

Sophisticated Agent Modeling: Language Design and Implementation for Large-Scale Simulation

José Eduardo Henriques

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Carlos Baquero-Moreno
External Examiner: Prof. Rui Gomes
Supervisor: Prof. Pedro Diniz

July 22, 2024

Abstract

This dissertation presents a comprehensive framework for analysing flow systems, through the integration between agent-based modelling and large-scale traffic simulations. The primary aim is to provide researchers with a scalable and flexible program for creating, observing, and evaluating traffic simulations. This framework leverages sophisticated machine learning methods, particularly deep reinforcement learning, to train agents in inferring behaviours for abstract scenarios. In doing so, it enables the exploration of dynamic interactions, emergent patterns, and potential bottlenecks within traffic networks, ultimately leading to optimized routing and re-routing strategies for flow systems.

Keywords: Agent Modelling, Multi-Agent Systems, MAS, Agent-Based Modelling, ABM, Large-Scale Simulation, Abstract Language Design, Traffic Network, Machine Learning, Reinforcement Learning, MARL, Flow Systems, TraCI, SUMO, Abstract Language

Resumo

Esta dissertação apresenta uma estrutura abrangente para analisar sistemas de fluxo, por meio da integração entre modelagem baseada em agentes e simulações de tráfego em larga escala. O objetivo principal é fornecer aos pesquisadores um programa escalável e flexível para criar, observar e avaliar simulações de tráfego. Esta estrutura aproveita métodos sofisticados de Machine Learning, particularmente Deep Reinforcement Learning, para treinar agentes na inferência de comportamentos para cenários abstratos. Ao fazer isso, ele permite a exploração de interações dinâmicas, padrões emergentes e potenciais engarrafamentos dentro de redes de tráfego, levando, em última análise, a estratégias otimizadas de roteamento e redirecionamento para sistemas de fluxo.

Keywords: Modelagem de Agentes, Sistemas Multiagentes, MAS, Modelagem Baseada em Agentes, ABM, Simulação em Grande Escala, Resumo Design de Linguagem, Rede de Tráfego, Aprendizado de Máquina, Aprendizado por Reforço, MARL, Sistemas de Fluxo, TraCI, SUMO, Linguagem Abstrata

Acknowledgements

First and foremost, I want to express my deepest appreciation for my family, namely my parents and my sister. Without their love, support, and faith in me, this dissertation wouldn't have been possible. They've always done what is beyond possible to ensure I have everything I need to achieve my goals and be who I want to be, so I'm excited and relieved that I finally achieved this milestone.

A gigantic thank you to all of my friends, for being there every time I needed them. Whether it was giving me words of encouragement or distracting me when I needed to unwind, they never failed to help me through this journey.

To my partner, girlfriend, wife, best friend, Greek sunshine, and the best person life has gifted me, Filippa, I owe a special spot in this text. Undoubtedly, the person who has put up with me the most because of my work. Someone who never failed to support me with her understanding and patience during the many late nights and stressful moments, and who ended up helping me so much. From the bottom of my heart, ευχαριστώ.

Last, but certainly not least, I would like to express my gratitude to my supervisors, Pedro Diniz and António Costa, for their time, guidance, and patience throughout this extended journey. I would also like to thank my previous supervisor, Rosaldo Rossetti, who, although is not my current supervisor, served a crucial role in the early stages of this work. Additionally, I would like to thank Giorgos Fragkiadakis, who helped me in a desperate moment and allowed me to make significant progress in this project.

José Eduardo Henriques

"Every day it gets a little easier...
But you gotta do it every day — that's the hard part.
But it does get easier."

Jogging Baboon, from BoJack Horseman

Contents

List of Figures

List of Tables

Abbreviations

ABM	Agent-Based Modelling
AMA	Agent-based Memetic Algorithm
ANN	Artificial Neural Networks
API	Application Programming Interface
AWS	Amazon Web Services
BN	Bayesian Networks
CES	Community Energy Systems
DRL	Deep Reinforcement Learning
DT	Decision Trees
EC	Evolutionary Computing
EI	Electronic Institutions
GP	Genetic Programming
GPI	Generalized Policy Iteration
GUI	Graphical User Interface
GVF	General Value Functions
IA2C	Independent Advantage Actor-Critic
IBM	Individual-Based Model
ITM	Intelligent Traffic Management
MA2C	Multi-Agent Advantage Actor-Critic
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent Simulation
MC	Monte Carlo
MDP	Markov Decision Process
ML	Machine Learning
OSM	OpenStreetMap
POMDP	Partially Observable Markov Decision Processes
PPO	Proximal Policy Optimization
PSR	Predictive State Representations
RF	Random Forest
RL	Reinforcement Learning
RLlib	Reinforcement Learning library
SUMO	Simulation of Urban MObility
TCP	Transmission Control Protocol
TD	Temporal-Difference
TRPO	Trust Region Policy Optimization
TraCI	Traffic Control Interface
VANET	Vehicular Ad-Hoc Networks
XGBoost	Extreme Gradient Boosting

Chapter 1

Introduction

1.1 Scope

In today's complex and interconnected world, optimizing flow systems has become crucial for ensuring efficiency and minimizing disruptions. Flow systems can be considered a network with a graph-like structure, where entities transit from one point to another, interacting with other entities and the environment in their travels. In these systems, there are usually several entities that perform different travels within a certain time period, often varying in origin, destination and frequency, thus naturally creating the need for efficient interaction between the entities to avoid underperformance of their travels. The ability to model and simulate such networks accurately is of great importance in developing effective strategies for routing and re-routing entities to optimize flow in these systems. The easiest and most practical type of flow system to imagine and consider would be a traffic network system.

Large-scale simulation of human behaviour in an urban setting is critical for the understanding of key planning and deployment issues. Results can help policymakers define urban development strategies, such as where to build more housing and of what type. Daily mobility studies and simulations can help them determine if more bus routes are needed and where the best public transportation transfer hubs would be located. In the context of risk analysis, these simulations could also provide key insights into the fragilities of the food supply network, thus allowing for changes that would increase its resiliency in case of natural disasters or conflicts.

1.2 Problem Statement and Motivation

With the recent exponential growth of populations in big cities across the globe, it has been recorded that an increasing number of vehicles on the roads has led to major traffic congestion problems in urban areas. Traditional approaches to traffic management have proven to be ineffective in dealing with these problems, and new methods are required to better manage traffic flow. Additionally, traffic networks pose unique challenges due to their intricate dynamics, diverse entities, and unpredictable nature.

1.3 Aim and Goals

Considering this, this research work focuses on designing sophisticated agent modelling in large-scale simulations to address the intricacies and complexities of flow systems, using traffic networks as a practical application. By employing practical examples and drawing insights from traffic management, this study explores how advanced simulation techniques can contribute to developing efficient routing and re-routing strategies.

The primary goal of this research is to examine how sophisticated agent modelling, in conjunction with large-scale simulations and machine learning, can enhance the efficiency of routing and re-routing strategies in traffic networks. A more realistic representation of the traffic ecosystem can be achieved by incorporating agent-based models that capture the decision-making processes inferred by machine learning methods instead of human-written hardcoded behaviours. This approach enables the exploration of dynamic interactions, emergent patterns, and potential bottlenecks within the system, leading to the identification of optimized routing solutions.

The final goal is to develop a scripting declarative language that allows for the definition of sophisticated agent behaviours that take into account a set of observable simulation variables and mission goals. These behaviours are from the traffic simulation practical application using reinforcement learning and are abstracted in order to be universally mapped into any other scenario application. This way, it is possible to ensure efficient decision-making for routing and re-routing for every entity across any flow system.

Additionally, by the end, an efficient and customizable template for creating, observing, and evaluating traffic simulations will be created.

1.4 Document Outline

This document contains sections dedicated to the topic's Background, the State of the Art, the adopted Methodology and Conclusions relating to the topic at hand.

1.5 Summary

In summary, this chapter has outlined the importance of optimizing flow systems, particularly in urban traffic networks, to enhance efficiency and reduce congestion. It identified the need for advanced simulation techniques and sophisticated agent modelling to address the challenges posed by traditional traffic management methods. The goals of this research include developing agent-based models and a declarative scripting language to improve routing strategies. The next sections will cover the background, state-of-the-art, methodology, and conclusions of this study.

Chapter 2

Background Knowledge

This chapter provides essential foundational concepts related to the topic of this dissertation. Likewise, it goes into detail on large-scale simulations, agent-based modelling, machine learning, and reinforcement learning. The chapter explains the core principles and methodologies of these fields, highlighting their significance and the challenges they address in complex system simulations, particularly in urban traffic networks.

2.1 Large-Scale Simulations

Large-scale simulations involve the modelling and simulation of complex systems at a large scale, providing a virtual environment to study their behaviour and dynamics. These simulations play a significant role across various domains by offering a powerful tool to understand, analyse, and optimize complex systems. From transportation and logistics to urban planning, healthcare, ecology, and social sciences, large-scale simulations enable researchers and decision-makers to explore different scenarios, test hypotheses, and evaluate the consequences of interventions. By replicating real-world conditions and interactions, these simulations provide valuable insights, inform decision-making processes, and facilitate the design of efficient strategies in a wide range of domains [27].

In the case of human behaviour in an urban setting, large-scale simulations are critical for understanding key planning and deployment issues [5]. Results from simulations covering daily mobility studies can help policymakers define urban development strategies, such as where to build more housing and of what type, and if more bus routes are needed and where the best public transportation transfer hubs would be located [14] [74]. In the context of risk analysis, these simulations could also provide key insights into the fragilities of the food supply network, thus allowing for changes that would increase its resiliency in case of natural disasters or conflicts [50].

Modelling and simulating large-scale systems pose several challenges due to their inherent complexity and the vast amount of data and interactions involved. One key challenge is the accurate representation of system components and their interactions, as the behaviour of entities at the individual level influences the system's overall behaviour. Incorporating the appropriate level of detail and granularity while considering computational limitations is crucial. Another challenge is data availability, as large-scale simulations often require significant amounts of real-world data for calibration and validation. Gathering, processing, and managing such extensive data can be time-consuming and resource-intensive. Additionally, scaling up simulations to handle large-scale systems requires efficient algorithms, high-performance computing infrastructure, and suitable simulation frameworks. Overcoming these challenges is essential to ensure the reliability, accuracy, and scalability of large-scale simulations, enabling researchers to gain meaningful insights and make informed decisions about complex systems [3] [10] [30] [64].

2.2 Agent-Based Modelling

Traditional modelling approaches often face limitations in capturing the complexities of large-scale systems. These approaches often rely on simplistic assumptions or aggregate-level representations, overlooking the heterogeneity and individual characteristics of entities within the system [43]. By assuming homogeneous behaviour or using average values, traditional models fail to capture the rich interactions, feedback loops, and non-linear behaviours that exist in real-world systems [11]. Moreover, traditional models may struggle to represent the adaptive and evolving nature of entities within the system, as they typically rely on fixed rules or assumptions that do not account for the dynamic decision-making processes and responses to changing conditions. These limitations can lead to inaccurate predictions and inadequate understanding of the complexities present in large-scale systems. Agent-based modelling addresses these limitations by explicitly modelling individual entities and their behaviours, allowing for a more nuanced and realistic representation of system dynamics [70].

Agent-based modelling is a powerful technique that represents entities within a system as autonomous agents with individual behaviours and interactions. In this approach, agents are endowed with specific characteristics, rules, and decision-making abilities, allowing them to interact with each other and their environment. Each agent operates based on its own internal state and external stimuli, leading to emergent patterns and system-level behaviours. By modelling entities as agents, agent-based modelling provides a bottom-up perspective, capturing the dynamics and interactions at the individual level that collectively shape the entire system's behaviour [77].

Incorporating realistic agent behaviours and decision-making processes in simulations is crucial for capturing the complexity and dynamics of real-world systems. By accurately representing the individual behaviours of agents, such as their preferences, goals, and decision rules, agent-based models can capture the diversity and complexity observed in real-world systems [27]. Realistic agent behaviours enable the exploration of how different strategies and actions at the individual level propagate through the system, leading to emergent patterns and system-wide effects [2]

[34]. Additionally, incorporating adaptive decision-making processes allows agents to respond dynamically to changing conditions, enabling simulations to capture the dynamic nature of complex systems and simulate realistic scenarios [73].

The paper "Synergistic Integration Between Machine Learning and Agent-Based Modelling: A Multidisciplinary Review" by Zhang et al. (2023) [81] provides a comprehensive review of the evolution of ABM throughout the years. In this work, the authors divide the progress of ABM in three periods:

- **From 1970 to 2000:** research on agent-based models (ABMs) primarily focused on natural life behaviours and characteristics. The term individual-based model (IBM) was more commonly used. Significant early works included Botkin et al.'s JABOWA model for forest succession [12], Shugart et al.'s gap model for forest dynamics [67], and Pacala et al.'s SORTIE model for predicting tree characteristics [51]. Studies also explored animal behaviours, such as fish flocking, and evolutionary computing approaches like agent-based memetic algorithms (AMAs) for optimization tasks. The introduction of swarm intelligence approaches further contributed to the understanding of decentralized systems.

- **From 2000 to 2010:** there was a noticeable shift in the adoption and application of ABMs. Around 2005, Grimm's paper [32] marked a turning point, popularizing the term agent-based model. The term multiagent system (MAS) also gained traction. ABMs expanded into human-related areas such as sociology, business, market analysis, land use science, and biological processes. Key applications included testing macro-sociological theories, business simulations, land-use pattern predictions, and modelling gene-protein interactions. This period saw the integration of new learning paradigms like particle swarm optimization and reinforcement learning, broadening the scope of ABM applications.

- **From 2010 to 2020:** ABMs became prevalent in fields such as social science, epidemiology, economics, and industrial business. Researchers applied ABMs to study complex, large-scale systems, including segregation patterns, HIV transmission dynamics, food risk assessment, and financial systems modelling. ABMs also facilitated the exploration of fiscal and monetary policies, industrial symbiosis networks, emergency evacuation plans, and household heating feedback. The terms ABM and MAS were used interchangeably in some contexts, particularly in computing domains, while the integration of machine learning techniques in ABMs gained momentum, enhancing the agents' decision-making and predictive capabilities.

2.3 Machine Learning

Machine learning is a branch of artificial intelligence that focuses on developing algorithms and statistical models that enable computer systems to learn and improve from experience without being explicitly programmed [47]. It is a computational approach that allows machines to automatically discover patterns, make predictions, and make data-based decisions [9]. Machine learning

algorithms analyse large datasets to identify underlying patterns, relationships, and trends. By utilizing techniques such as supervised learning, unsupervised learning, and reinforcement learning, machine learning algorithms can learn from labelled or unlabelled data, recognize complex patterns, and generalize their knowledge to make accurate predictions or take appropriate actions in new situations [31]. It finds applications in various domains, including image and speech recognition, natural language processing, recommendation systems, predictive analytics, and autonomous decision-making [63].

Machine learning in agent-based modelling has emerged as a powerful approach to enhance the realism and effectiveness of simulations [11]. Machine learning techniques like neural networks, decision trees, and reinforcement learning algorithms can infer agent behaviours, learn from data, and adapt to changing environments [69]. Researchers can capture complex and realistic agent behaviours that evolve over time by integrating machine learning into agent-based models [46]. Machine learning algorithms can leverage real data or simulated environments to train agents, allowing them to learn optimal decision-making strategies, adapt to new scenarios, and interact more realistically with their environment and other agents [52].

2.4 Reinforcement Learning

This section covers concepts and methodologies from "Reinforcement Learning: An Introduction" by Richard S. Sutton and Andrew G. Barto [69]. It provides an overview of the reinforcement learning problem, finite Markov decision processes, dynamic programming techniques, and recent advancements in the field.

2.4.1 The Reinforcement Learning Problem

Reinforcement learning (RL) is a branch of machine learning where an agent learns to make decisions by taking actions in an environment to maximize cumulative reward. Unlike supervised learning, where the agent is provided with explicit instructions on what actions to take, RL requires the agent to discover the best actions through trial and error. This process involves the agent interacting with its environment, where its actions influence future states and rewards. Key characteristics of RL problems include their closed-loop nature, the agent's need to explore actions, and the extended consequences of actions over time. Reinforcement learning systems typically comprise four main elements:

- **The Policy:** The agent's strategy to determine actions based on the current state.
- Reward Signal: The feedback received from the environment in response to the agent's actions defines the goal of the RL problem.
- **Value Function:** An estimate of the total future rewards obtained from each state, guiding the agent to make decisions that maximize long-term rewards.

- **Model of the Environment (optional):** A representation of the environment used for planning by predicting future states and rewards based on current actions.

While RL methods often involve estimating value functions, it is not the only approach. Evolutionary methods, for instance, evaluate the performance of different policies over many episodes without learning during individual interactions. RL focuses on learning from interactions with the environment, which can be more efficient than evolutionary methods in many cases. Policy gradient methods, which adjust policy parameters directly based on performance feedback, are another approach within RL that does not rely on value functions.

The history of RL is rooted in two main threads: trial-and-error learning from psychology and optimal control from engineering. The trial-and-error approach, influenced by Thorndike's Law of Effect, emphasizes learning from the consequences of actions. Optimal control, formalized by Bellman's dynamic programming and Markov decision processes (MDPs), focuses on minimizing cost functions over time. These threads converged in the late 1980s, integrating dynamic programming with online learning to form modern RL.

2.4.2 Finite Markov Decision Processes

The reinforcement learning problem is conceptualized as an interaction between an agent and its environment. The agent makes decisions by selecting actions based on the current state of the environment. The environment responds to these actions by transitioning to new states and providing rewards. This interaction is cyclical and occurs over discrete time steps $t = 0, 1, 2, 3, \ldots$. At each time step, the agent receives a state S_t from the environment, chooses an action A_t from a set of available actions $A(S_t)$, and then receives a reward R_{t+1} and a new state S_{t+1} from the environment.

The agent's goal in reinforcement learning is to maximize the cumulative reward it receives over time. The reward signal is a scalar feedback signal indicating how well the agent performs at each time step. The agent's objective is to find a policy, a mapping from states to actions, that maximizes the expected sum of the rewards over the future.

The return G_t is the total accumulated reward from time step t onward. For episodic tasks, which have a natural end, the return is the sum of the rewards until the end of the episode. For continuing tasks, the return is typically defined using a discount factor γ, ensuring that the rewards' sum is finite. The return is then given by:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

where $0 \leq \gamma < 1$. This discount factor reflects the idea that immediate rewards are more valuable than distant future rewards.

To handle both episodic and continuing tasks within a unified framework, the return G_t is used as the standard measure of future rewards. For episodic tasks, the return naturally ends when the

episode terminates. For continuing tasks, the discount factor γ ensures that the sum converges. This unified notation allows for a consistent treatment of different types of tasks.

A state signal is said to have the Markov property if it captures all relevant information about the past that is necessary to predict the future. Formally, a state S_t is Markov if:

$$\Pr(S_{t+1} = s' \mid S_t = s, A_t = a) = \Pr(S_{t+1} = s' \mid S_0, A_0, S_1, A_1, \ldots, S_t = s, A_t = a) \tag{2.2}$$

This implies that the future is conditionally independent of the past, given the present state. Environments that satisfy this property are called Markov decision processes (MDPs). The Markov property is crucial because it simplifies the decision-making process, allowing the agent to base its decisions solely on the current state.

A finite Markov decision process (MDP) is defined by its finite set of states S, a finite set of actions A, and the one-step dynamics of the environment specified by the transition probability $p(s', r|s, a)$. This probability denotes the likelihood of transitioning to state s' and receiving reward r given that the agent is in state s s and takes action a. The dynamics of a finite MDP are fully characterized by these probabilities.

Value functions are used to estimate the expected return from each state (or state-action pair) under a particular policy. The state-value function $v_\pi(s)$ gives the expected return when starting from state s and following policy π:

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s\right] \tag{2.3}$$

The action-value function $q_\pi(s,a)$ gives the expected return starting from state s, taking action a, and thereafter following policy π:

$$q_\pi(s,a) = E_\pi[G_t \mid S_t = s, A_t = a] \tag{2.4}$$

These functions provide a basis for evaluating and improving policies.

Optimal value functions define the best possible performance that can be achieved from each state (or state-action pair). The optimal state-value function $v_*(s)$ is the maximum expected return achievable from state s):

$$v_*(s) = \max_\pi v_\pi(s) \tag{2.5}$$

Similarly, the optimal action-value function $q_*(s,a)$ is the maximum expected return achievable from state s and action a:

$$q_*(s,a) = \max_{\pi} q_\pi(s,a) \tag{2.6}$$

These optimal functions satisfy the Bellman optimality equations, which can be solved to find the optimal policy.

In practice, finding exact solutions to the Bellman equations for large-scale MDPs is often infeasible due to the computational complexity. Therefore, approximation methods are used to find near-optimal solutions. These methods include various forms of function approximation and iterative algorithms that converge to optimal or near-optimal policies over time.

2.4.3 Dynamic Programming

Policy evaluation is the process of determining the value function v_π for a given policy π. The value function represents the expected return starting from a state s and following the policy π. For a finite Markov Decision Process (MDP), the value function $v_\pi(s)$ satisfies the Bellman equation:

$$v_\pi(s) = E_p i[G_t|S_t = s] = \sum_{a\in A} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \tag{2.7}$$

This equation can be solved iteratively, starting with an initial guess v_0 and repeatedly applying the update rule:

$$v_{k+1}(s) = \sum_{a\in A} \pi(a|s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')] \tag{2.8}$$

This iterative process continues until the values converge.

Once the value function v_π for a policy π is known, the policy can be improved by acting greedily with respect to v_π. This means selecting actions that maximize the expected return based on the current value function. The improved policy π' is given by:

$$\pi'(s) = \arg\max_{a\in A} \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \tag{2.9}$$

If the new policy π' is different from the old policy π, the process can be repeated with π' replacing π. This iterative procedure is known as policy iteration.

Policy iteration involves two steps: policy evaluation and policy improvement. These steps are repeated iteratively until the policy converges to the optimal policy π_*. The algorithm for policy iteration can be summarized as follows:

1. Initialize a policy π.

2. Evaluate the current policy π to obtain π'.

3. Improve the policy to obtain a new policy π'.

4. If $\pi' \neq \pi$, set $\pi \leftarrow \pi$ and repeat from step 2; otherwise, terminate.

Policy iteration converges to the optimal policy and optimal value function because each policy improvement step is guaranteed to produce a strictly better policy unless the current policy is already optimal.

Value iteration is an alternative to policy iteration that combines policy evaluation and improvement into a single step. Instead of evaluating a policy to completion, value iteration updates the value function directly using the Bellman optimality equation:

$$v_{k+1}(s) = \max_{a \in A} \sum_{s',r} p(s',r|s,a)[r + \gamma v_k(s')] \tag{2.10}$$

This update rule is applied iteratively for all states until the value function converges. Once the value function v_* converges, the optimal policy π_* can be derived by acting greedily with respect to v_*:

$$\pi_*(s) = \arg\max_{a \in A} \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] \tag{2.11}$$

Value iteration is often preferred over policy iteration because it can converge faster in practice, particularly for large state spaces.

Asynchronous dynamic programming methods update the value function for some states at each iteration, rather than all states simultaneously. This flexibility allows for more efficient use of computational resources and can lead to faster convergence in practice. Asynchronous methods can be particularly useful in large or complex MDPs where updating all states simultaneously is computationally infeasible.

Generalized Policy Iteration (GPI) refers to the interplay between policy evaluation and policy improvement, where both processes occur simultaneously or in a more flexible sequence. The core idea is that the value function and policy are continually updated towards optimality, efficiently leveraging any available computational resources. GPI encompasses policy iteration and value iteration as special cases, providing a unifying framework for understanding dynamic programming methods in reinforcement learning.

Dynamic programming methods, while powerful, can suffer from the "curse of dimensionality," where the state space grows exponentially with the number of state variables. To address this, various techniques, such as function approximation, state aggregation, and hierarchical decomposition, are employed to make dynamic programming feasible for large-scale problems. These methods reduce computational complexity and enable the application of dynamic programming to real-world problems.

2.4.4 Monte Carlo

Monte Carlo (MC) methods are a class of algorithms that rely on repeated random sampling to obtain numerical results. In the context of reinforcement learning, Monte Carlo methods are used to estimate the value of states or state-action pairs based on empirical returns observed during interaction with the environment.

The value of a state $v_\pi(s)$ under policy π can be estimated using the sample average of the returns following visits to that state:

$$v_\pi(s) \approx \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_i \tag{2.12}$$

where $N(s)$ is the number of times state s has been visited and G_i is the return following the i-th visit to s. This approach requires complete episodes to be observed, as the returns $G_i$are calculated based on the total rewards received from the time of the visit to the end of the episode.

For estimating the value of state-action pairs $q_\pi(s,a)$, the Monte Carlo method involves averaging the returns following visits to the state-action pair (s,a):

$$q_\pi(s,a) \approx \frac{1}{N(s,a)} \sum_{i=1}^{N(s,a)} G_i \tag{2.13}$$

where $N(s,a)$ is the number of times action a has been taken in state s, and G_i is the return following the i-th occurrence of (s,a). This allows for the evaluation of policies based on empirical data without requiring knowledge of the environment's dynamics.

Monte Carlo control methods aim to improve the policy by combining policy evaluation and policy improvement steps iteratively. The overall process involves:

1. **Policy Evaluation:** Using Monte Carlo prediction to estimate $q_\pi(s,a)$ for the current policy π

2. **Policy Improvement:** Updating the policy to be greedy with respect to the current action-value estimates:

$$\pi'(s) = \arg\max_a q_\pi(s,a) \tag{2.14}$$

This iterative process continues until the policy converges to an optimal policy.

Exploring starts ensure that all state-action pairs are explored sufficiently by starting each episode in a randomly selected state-action pair. However, this is often impractical. An alternative approach is to use ε-greedy policies, where actions are chosen randomly with a small probability ε and greedily with probability $1-\varepsilon$. This method ensures sufficient exploration while gradually improving the policy.

Monte Carlo methods can also be applied off-policy, where the target policy π differs from the behaviour policy b used to generate data. Importance sampling corrects for the difference between the policies by weighting returns by the ratio of the target to behaviour policy probabilities:

$$v_\pi(s) = E[\rho_t G_t \mid S_t = s] \tag{2.15}$$

where

$$\rho_t = \prod_{k=t}^{T-1} \frac{\pi(A_k \mid S_k)}{b(A_k \mid S_k)} \tag{2.16}$$

is the importance sampling ratio.

Monte Carlo methods can be implemented incrementally, allowing for updates to value estimates after each step, rather than waiting for the end of an episode. This is particularly useful for environments with long episodes or for continuing tasks. The incremental update rule for state values is:

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) \tag{2.17}$$

where α is a step-size parameter.

Off-policy Monte Carlo control uses importance sampling to learn the value of the target policy π while following a different behaviour policy b. The key is to apply importance sampling to both policy evaluation and policy improvement steps to derive an optimal target policy.

2.4.5 Temporal-Difference Learning

Temporal-Difference (TD) learning is a class of model-free reinforcement learning methods that learn by bootstrapping from the current estimate of the value function. Unlike Monte Carlo methods, which wait until the end of an episode to update value estimates, TD learning updates estimates based on partial returns observed after each step. The simplest TD method is TD(0), which updates the value of the current state S_t based on the observed reward R_{t+1} and the estimated value of the next state $V(S_t + 1$:

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.18}$$

where α is the learning rate, and γ is the discount factor.

TD learning combines the benefits of Monte Carlo and dynamic programming methods. It can learn directly from raw experience without a model of the environment, like Monte Carlo methods, and it updates estimates based on other learned estimates, as in dynamic programming.

This allows TD methods to learn efficiently and quickly, even from incomplete episodes. Key advantages include:

- **Sample Efficiency:** TD methods update value estimates at every step, making them more sample-efficient than Monte Carlo methods.
- **Online Learning:** TD methods can learn from each interaction step-by-step, making them suitable for online learning scenarios.
- **Low Variance:** TD methods typically exhibit lower variance in their estimates compared to Monte Carlo methods by bootstrapping from the current value function.

TD(0) converges to the true value function V_π for any fixed policy π under certain conditions, such as using a decreasing step-size parameter and ensuring that all states are visited infinitely often. The theory of stochastic approximation and dynamic programming guarantees this convergence.

Sarsa is an on-policy TD control algorithm that updates the action-value function Q based on the state-action pair, the reward, the next state, and the next action chosen according to the current policy. The update rule for Sarsa is:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1},a) - Q(S_t,A_t)] \tag{2.19}$$

This algorithm ensures that the policy being improved is the same as the policy used to generate the data, making it suitable for learning in dynamic and uncertain environments.

6.5 Q-learning: Off-policy TD Control Q-learning is an off-policy TD control algorithm aiming to learn the optimal action-value function Q_*, directly approximating the maximum expected return. The updated rule for Q-learning is:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)] \tag{2.20}$$

Unlike Sarsa, Q-learning uses the maximum estimated value of the next state rather than the value of the next action taken by the current policy. This makes Q-learning more flexible, but also introduces potential instability due to overestimation biases.

6.6 Expected Sarsa Expected Sarsa is a variant of Sarsa that uses the expected value of the next state's action-value function, rather than the actual next action taken. The update rule for Expected Sarsa is:

$$Q(S_t,A_t) \quad \leftarrow \quad Q(S_t,A_t) + \alpha \left[R_{t+1} + \gamma \sum_a \pi(a \mid S_{t+1}) Q(S_{t+1},a) - Q(S_t,A_t) \right] \tag{2.21}$$

This approach reduces variance compared to regular Sarsa and Q-learning by averaging over the policy's action probabilities.

Maximization bias in Q-learning refers to the tendency to overestimate the value of the maximum action due to noise in the value estimates. Double Q-learning addresses this issue by using two separate value functions to decouple the action selection from the value estimation:

$$Q_1(S_t,A_t) \leftarrow Q_1(S_t,A_t) + \alpha \left[R_{t+1} + \gamma Q_2(S_{t+1}, \arg\max_a Q_1(S_{t+1},a)) - Q_1(S_t,A_t)\right] \tag{2.22}$$

$$Q_2(S_t,A_t) \leftarrow Q_2(S_t,A_t) + \alpha \left[R_{t+1} + \gamma Q_1(S_{t+1}, \arg\max_a Q_2(S_{t+1},a)) - Q_2(S_t,A_t)\right] \tag{2.23}$$

Double Q-learning mitigates the overestimation bias by alternating updates between Q_1 and Q_2.

2.4.6 Frontiers

Initially, value functions were tied to specific reward predictions and policies. Over time, these have been expanded to include off-policy learning and state-dependent discount functions, allowing for more nuanced and flexible predictions.

A key advancement discussed is the idea of general value functions (GVFs), which extend beyond rewards to predict various signals, such as sensory inputs or internal signals. These predictions can be formulated as:

$$v_{\pi,\gamma,C}(s) = E\left[\sum_{k=t}^{\infty}\left(\prod_{i=t+1}^{k}\gamma(S_i)\right)C_{k+1} \mid S_t = s, A_{t:\infty} \sim \pi\right] \tag{2.24}$$

where C_t is the cumulant signal, representing the signal being predicted, and $\gamma(S_i)$ is the state-dependent discount function. GVFs offer a way to make diverse predictions about the environment, enhancing the agent's ability to model and interact with its surroundings effectively.

Temporal abstraction allows reinforcement learning agents to operate at different levels of granularity, from low-level actions to extended courses of action spanning many time steps. This is formalized through the concept of options, which are pairs consisting of a policy and a termination function. An option $\omega = (\pi_\omega, \beta_\omega)$ is executed until the termination condition β_ω is met. This framework enables hierarchical reinforcement learning, where the agent can select extended actions (options) that persist over multiple time steps, thus simplifying complex decision-making processes. The value function for options can be expressed similarly to standard value functions but incorporates the expected cumulative reward and state transitions associated with the option.

$$q_\pi(s,\omega) = E\left[\sum_{k=0}^{\tau-1}\gamma^k r_{t+k} + \gamma^\tau v_\pi(s_{t+\tau}) \mid S_t = s, \omega_t = \omega\right] \tag{2.25}$$

where τ is the termination time of the option.

In many realistic scenarios, the agent cannot fully observe the environment's state. One approach to deal with this is to use belief states in Partially Observable Markov Decision Processes (POMDPs), where the agent maintains a probability distribution over possible states given its observations. Another approach is Predictive State Representations (PSRs), where the state is represented as a vector of predictions about future observations. This method leverages observable data to ground the agent's state representation, making it more practical for learning and planning in partially observable environments.

Designing effective reward signals is critical to guiding reinforcement learning agents towards desirable behaviours. However, poorly designed rewards can lead to unintended and possibly harmful outcomes. The chapter emphasizes the importance of crafting reward signals that align well with the overall goals and constraints of the system, considering both immediate and long-term effects. One strategy is to use shaping, where the reward signal is adjusted over time to guide the agent towards the desired behaviour gradually. Another method involves using imitation learning or inverse reinforcement learning to derive reward signals from observing expert behaviour. Several unresolved challenges and areas for future research in reinforcement learning consist of:

- Developing scalable and efficient planning methods with learned environment models.
- Automating the choice of auxiliary tasks and GVFs to enhance the agent's learning process.
- Ensuring safe and robust learning in real-world environments, which involves risk management and adherence to best practices in control engineering.
- Addressing the limitations of current deep learning methods, particularly their suitability for online, incremental learning settings.
- Exploring the interaction between behaviour and learning through computational curiosity, where intrinsic rewards are used to drive exploration and learning.

2.5 Summary

This chapter explores the components necessary for understanding and developing advanced simulation systems. It begins with an overview of large-scale simulations and their application across various domains, emphasizing their importance in urban planning and risk analysis. The chapter then discusses agent-based modelling, illustrating how it overcomes the limitations of traditional models by representing entities as autonomous agents with individual behaviours. It also reviews the integration of machine learning in agent-based models, enhancing their realism and adaptability. Finally, the chapter covers reinforcement learning, detailing its problem space, the Markov decision process framework, and dynamic programming techniques, underscoring their relevance in optimizing decision-making processes within simulations.

Chapter 3

State of the Art

This chapter serves as a comprehensive exploration of the current landscape of research and developments relevant to the proposed work. This section aims to provide a thorough understanding of existing theories and methodologies that form the foundation for the decided methodology presented in the next chapter.

3.1 Machine Learning in Agent-Based Modelling

This section heavily focuses on the study carried out by Zhang et al. (2023) in their work "Synergistic Integration Between Machine Learning and Agent-Based Modelling: A Multidisciplinary Review"[81]. This work provides a comprehensive review of the integration of ML and ABM, addressing various scenarios and their related algorithms, frameworks, procedures, and multidisciplinary applications.

The introduction of this multidisciplinary review paper sets the stage by highlighting the growing importance of integrating Machine Learning (ML) and Agent-Based Modeling (ABM) across various fields. The synergy between ML and ABM offers a powerful framework for simulating complex systems and improving decision-making processes. The review aims to provide a comprehensive overview of how ML can enhance ABM, addressing the challenges and opportunities that arise from this integration. The paper begins by defining the key concepts of ML and ABM. Machine Learning is described as a subset of artificial intelligence that focuses on developing algorithms that allow computers to learn from and make predictions based on data. Agent-Based Modeling, on the other hand, is a computational approach used to simulate the actions and interactions of autonomous agents to assess their effects on the system as a whole. The authors emphasize the significance of this integration in various domains, such as traffic management, healthcare, economics, and environmental science. By leveraging the strengths of both ML and ABM, researchers can create more accurate and adaptable models that better reflect real-world complexities. The introduction also outlines the structure of the paper, which includes a detailed

review of the background and trends of ABM, the integration of ML with ABM, and the future directions of this interdisciplinary research.

3.1.1 Scenarios of Applying ML to ABM

The integration of Machine Learning (ML) with Agent-Based Models (ABM) can be implemented in various scenarios to enhance model functionality and accuracy. Firstly, ML algorithms can be broadly categorized into four types:

- **Supervised learning:** is a type of ML where the model is trained using labelled data, meaning that each training example is paired with an output label. The primary objective is to learn a mapping from inputs to outputs that can be used to predict the labels for new, unseen data. This approach is widely used for classification and regression tasks. For instance, in ABMs, supervised learning can be applied to predict agent behaviours or environmental outcomes based on historical data, improving the accuracy and reliability of simulations.

- **Unsupervised learning:** unlike supervised learning, unsupervised learning involves training a model on data without labelled responses. The goal is to identify hidden patterns or intrinsic structures within the data. This approach is particularly useful for clustering, anomaly detection, and association tasks. Within ABMs, unsupervised learning can help discover group behaviours or patterns among agents that were not predefined, thereby revealing insights that might be overlooked using traditional methods.

- **Semisupervised learning:** bridges the gap between supervised and unsupervised learning by using a small amount of labelled data along with a large amount of unlabelled data. This method is especially beneficial when labelling data is expensive or time-consuming. It can significantly improve model performance by leveraging both labelled and unlabelled data. This approach can enhance the model's predictive capabilities in ABM contexts by using available labelled data to guide the learning process while exploring the broader dataset.

- **Reinforcement learning (RL):** is a type of ML where an agent learns to make decisions by performing actions in an environment to maximize cumulative reward. RL is particularly effective in dynamic and complex environments where the agent must learn from interactions and adapt its strategy over time. In ABMs, RL can be used to model adaptive behaviours and decision-making processes, allowing agents to optimize their actions based on feedback from the environment. This iterative learning process enables the development of more sophisticated and realistic simulations.

Each type of ML algorithm brings unique strengths to the integration with ABMs, providing a robust framework for simulating and analysing complex systems. Researchers and practitioners can create more accurate, dynamic, and insightful agent-based models by understanding and leveraging these algorithms.

Four primary scenarios illustrate the application of ML in ABM:

1. **Microagent Situational Awareness Learning:** involves using ML to enhance the awareness of individual agents about their environment. ML algorithms predict relevant social, economic, or environmental variables or agent-related behaviours in this scenario. Both supervised learning and reinforcement learning (RL) algorithms are employed, depending on the clarity and labelling of the objectives. Supervised learning is used when the outcomes can be clearly defined and labelled, while RL is preferred for more complex behaviour dynamics where direct labelling is not feasible.

2. **Microagent Behaviour Interventions:** focuses on modifying agents' behaviour based on ML-driven insights. This approach uses ML to infer improved decision-making models from empirical and real-time data. Supervised ML algorithms, such as decision trees (DT), Bayesian networks (BN), and artificial neural networks (ANN), can classify and select optimal actions for agents. In contrast, RL can incorporate streaming data to adapt behaviours over time. This scenario highlights how ML can empower agents to achieve their goals more effectively by continually optimizing their actions based on new data.

3. **Macrolevel Emergence Emulator:** utilizes ML to map the relationship between microagent parameters and macrolevel system outcomes. This approach addresses the "dimensional disaster" by building emulators that predict macro-level emergent properties from micro-level behaviours. Techniques such as extremely boosted gradient trees (XGBoost) and random forests (RF) are used to explore parameter spaces and predict system outcomes. This scenario is essential for understanding and predicting complex emergent phenomena arising from numerous agents' interactions.

4. **Macro ABMs Decision-Making:** In Macro ABMs Decision-Making, ML algorithms assist macrolevel policymakers in making informed decisions that influence microagent behaviours. This scenario treats the policymaker as a single "macroagent" whose decisions affect the entire system. RL algorithms, such as Q-learning, are commonly used to reinforce decision-making processes. This approach is applied in various fields, including traffic signal control, land-use planning, and healthcare, where macro-level decisions need to guide the behaviour of numerous agents to achieve specific objectives.

Each scenario leverages ML techniques to address specific aspects of agent behaviour and system-level outcomes, demonstrating the versatility and power of combining these two methodologies. These scenarios illustrate the multifaceted roles that ML can play in enhancing the functionality and accuracy of ABMs. By integrating ML techniques, ABMs can become more adaptive, predictive, and effective in simulating complex systems and informing decision-making processes.

3.1.2 Multidisciplinary Review for Roles of ML in ABM

- **A. Microagent Situational Awareness Learning**

1. **Supervised ML-Based Method:** Supervised Machine Learning methods significantly enhance the situational awareness of microagents in Agent-Based Modelling (ABM). These methods involve training models on labelled datasets to enable agents to predict and respond to various environmental conditions and behavioural patterns.
   - **a) Situational Awareness Learning of Environment:** Supervised ML methods enhance microagent situational awareness by predicting environmental variables crucial to agent decision-making. For example, supervised algorithms can predict weather conditions or market trends, providing agents with the data needed to make informed decisions. By processing historical data, these methods can identify patterns and trends that agents can use to anticipate changes in their environment, thereby improving their responsiveness and adaptability.
   - **b) Agent-Related Behavioural Prediction:** Supervised ML techniques also excel in predicting agent-related behaviours. These algorithms can analyse past actions and outcomes to forecast future agent behaviours. This capability is particularly useful in applications such as traffic management, where predicting individual driver behaviours can improve overall system efficiency. By learning from labelled datasets, these methods can provide accurate predictions that help agents adjust their strategies proactively.
2. **RL-Based Method:** Reinforcement Learning (RL) methods are employed when the behavioural dynamics of agents cannot be clearly defined or labelled. RL algorithms enable agents to learn optimal behaviours through interactions with their environment. This learning process involves receiving feedback from actions and continuously improving strategies based on rewards and penalties. RL is particularly effective in dynamic environments where agents need to adapt to changes and learn from real-time experiences.

- **B. Microagent Behavioural Intervention**

1. **Agent Behavioural Interventions With Supervised ML:** Supervised ML algorithms facilitate behavioural interventions by providing empirical data that agents can use to optimize their actions. Techniques like decision trees (DT), Bayesian networks (BN), and artificial neural networks (ANN) can classify different actions and determine the most effective ones for achieving desired outcomes. These interventions enable agents to refine their decision-making processes, leading to more efficient and effective behaviours.
2. **Agent Behavioural Interventions With RL:** Reinforcement Learning (RL) is particularly effective for behavioural interventions, as it allows agents to learn and adapt their behaviours based on feedback from the environment. RL-based interventions can be tailored to different tasks, including cooperative, competitive, and mixed tasks.

  - **a) Cooperative Tasks:** In cooperative tasks, RL algorithms help agents work together to achieve common goals. Agents learn to coordinate their actions to maximize collective rewards. This is particularly useful in scenarios like multi-robot systems or collaborative filtering, where agents must collaborate to complete tasks more efficiently than they could individually.
  - **b) Competitive Tasks:** For competitive tasks, RL algorithms enable agents to develop strategies that optimize their performance against opponents. By simulating competitive environments, agents learn to anticipate and counteract the actions of others, improving their chances of success. Applications include economic markets and strategic games where competitive interactions are prevalent.
  - **c) Mixed Tasks:** Mixed tasks involve both cooperative and competitive elements, requiring agents to balance collaboration and competition. RL algorithms help agents navigate these complex interactions by learning when to cooperate and when to compete. This versatility is crucial in real-world scenarios like traffic systems, where drivers must cooperate to avoid accidents while competing for optimal routes.

- **C. Macro ABMs/Emergence Emulator:** The Macrolevel Emergence Emulator scenario leverages ML to model the emergence of macro-level patterns from micro-level interactions. Techniques such as extremely boosted gradient trees (XGBoost) and random forests (RF) are employed to predict macro-level phenomena based on micro-level agent behaviours. This approach helps in understanding and forecasting complex system-wide behaviours that emerge from the interactions of numerous agents, providing valuable insights into the dynamics of large-scale systems.

- **D. Macrolevel ABMs Decision-Making:** In Macrolevel ABMs Decision-Making, ML algorithms support policymakers in making strategic decisions that influence microagent behaviours. Reinforcement learning (RL) algorithms, such as Q-learning, are used to optimize policy decisions that affect the entire system. This approach is applied in various domains, including urban planning, healthcare, and environmental management, where decisions at the macro level need to guide and shape the behaviours of numerous individual agents to achieve overarching objectives.

3.2 Traffic Simulations

3.2.1 SUMO (Simulation of Urban MObility)

SUMO [38] is an open-source, flexible, and extensible platform designed to model and simulate the detailed behaviour of individual vehicles. The primary objective of SUMO is to assist in analysing and optimizing traffic systems by providing a realistic and dynamic simulation environment. The introduction highlights the key features of SUMO, including its capability to handle

large-scale networks, support for various types of transportation modes, and interoperability with other software tools. Lopez et al. (2018) explain in great detail the intricacies of SUMO in their paper "Microscopic Traffic Simulation using SUMO" [42].

In SUMO, users can import existing road networks from various data sources or create custom networks using SUMO's network editor tools. This step involves defining road layouts, intersections, traffic signals, and other infrastructure elements. Traffic demand is generated based on historical data, synthetic models, or user-defined parameters. This step involves specifying the types and numbers of vehicles, their routes, and departure times. Users configure the simulation parameters, including simulation duration, time steps, and output options. This stage ensures that the simulation environment accurately reflects the intended study parameters. Users can monitor the simulation in real time, adjust parameters, and collect data on various traffic metrics. SUMO also provides tools for analysing and visualizing the simulation results.

SUMO allows for the creation of a detailed road network representation, including road segments, lanes, intersections, traffic signals, and other relevant infrastructure components. Users can import networks from external sources such as OpenStreetMap (OSM), which allows for the virtual representation of real life scenarios. Using SUMO's network editor, it is possible to set up traffic signal programs, including signal phases, timings, and coordination between multiple signals, allowing for both fixed-time and adaptive signal control strategies. It also allows for the definition of lane attributes like lane width, speed limits, and lane restrictions. Incorporating other network elements like pedestrian crossings, bus stops, and parking areas to create a comprehensive traffic environment ensures that the simulation accurately reflects real-world driving conditions

SUMO allows for generating trips based on various models, including random distributions, activity-based models, and real-world data. Routes are defined by specifying the origin, destination, departure time, and mode of transport for each trip. SUMO uses dynamic route assignment algorithms that can adapt to changing traffic conditions, adapting traffic flows based on real-time simulation data. This includes re-routing vehicles in response to congestion, accidents, or other disruptions, which helps to manage traffic more effectively by dynamically adjusting routes. Integrating different modes of transport, such as cars, buses, bicycles, and pedestrians, to simulate a multi-modal transportation system provides a complete view of the traffic network and its interactions.

Coupling SUMO with other simulation tools to create an interactive traffic simulation environment is possible through TraCI (Traffic Control Interface). TraCI communicates with SUMO through a TCP socket, being able to receive data about individual simulation components and directly modify them in real time.

3.2.2 TraCI (Traffic Control Interface)

TraCI [76] is an API created for interlinking road traffic and network simulators, allowing for an effective evaluation of the impact of vehicular applications on traffic patterns. Unlike traditional methods that rely on static mobility traces, TraCI allows for real-time coupling, enabling the dynamic adaptation of driver behaviour during simulation runtime. This approach improves

the accuracy of simulations and supports the systematic evaluation of VANET (Vehicular Ad-Hoc Networks) applications under realistic conditions.

The concept of mobility primitives is introduced to decompose complex mobility patterns into basic commands such as 'change speed', 'change lane', and 'change route'. These primitives are independent of VANET applications and depend on macroscopic and microscopic mobility constraints. The system architecture of TraCI involves concurrent operation of the road traffic and network simulators, connected via a TCP-based data exchange protocol. The road traffic simulator is extended with a TraCI-Server, while the network simulator operates with a TraCI-Client. This setup allows the network simulator to control the traffic simulator, enabling real-time adjustments to vehicle movements based on VANET application outputs. Data exchange is managed through periodic commands from the network simulator, ensuring synchronized simulation steps between the two simulators.

3.3 Multi-Agent Systems in Traffic Simulations

Modern cities are experiencing unprecedented growth in population and financial power, leading to an increase in private vehicle ownership and reliance on urban road networks [71]. The resulting traffic volumes have exceeded the capacity of existing infrastructure, making effective traffic management crucial. Traditional solutions like building new roads or adding more lanes to existing ones have been proven ineffective [78] [26], often leading to phenomena like Braess' paradox [13]. Current traffic control systems are often static, relying on past data rather than real-time information, which is insufficient in modern, chaotic traffic environments. Therefore, there is a need to understand traffic scenarios and create adaptive and intelligent traffic management systems that can learn and respond to real-time conditions [72].

Intelligent Traffic Management (ITM) can be achieved using Multi-Agent Systems (MAS) in combination with traffic simulations. In this approach, each traffic component acts as an autonomous and adaptive agent. These agents can communicate and cooperate to optimize traffic flow locally and globally, leading to network throughput improvements. It allows for real-time interaction with simulators, enabling the creation and evaluation of MAS-based traffic management solutions. This is of special interest to urban planners since it provides insights and possible solutions without the high costs associated with modifying road infrastructure. It does not rely on physical components of the real world like self-driving cars [16] or the usage of computer vision [57] [25]. This approach also saw an increase in popularity due to advancements in AI and computational power [80], which allowed for the modelling of complex systems with numerous interacting sophisticated agents.

There are several different applications of agent-based technology in traffic systems [22]. One of the most natural and popular approaches consists of treating each vehicle as an agent, focusing on their decision-making behaviours for [23] [49] [61] [62]. Another approach simply focuses on the prediction of congestion in traffic networks [59] [79].

3.4 Reinforcement Learning in Traffic Simulations

Reinforcement Learning (RL) is a subset of machine learning where an agent learns to make decisions by performing actions in an environment to maximize cumulative reward. In the context of traffic simulations, RL involves agents (vehicles or traffic signals) learning optimal policies to improve traffic flow and reduce congestion. In RL, the agent receives feedback from the environment through rewards or penalties based on its actions and, over time, adjusts its strategy to achieve the highest possible reward. This method is particularly effective in dynamic and complex environments such as urban traffic networks, where traditional rule-based systems may fall short. Using RL in traffic simulations provides a virtual environment where it is possible to develop and test adaptive traffic management strategies, allowing for real-time optimization of traffic flow and dynamic response to changing conditions. Because of this, several studies have been conducted using the methodology in this field.

3.4.1 Curriculum Learning

One of the significant challenges in reinforcement learning is dealing with sparse reward signals. Sparse rewards occur when the agent receives feedback infrequently, making it difficult to learn optimal policies efficiently. This issue is prevalent in traffic simulations, where meaningful feedback (such as significant congestion reduction) may not occur frequently. Makri et al. (2023) tackle this issue in their article "Curriculum based Reinforcement Learning for traffic simulations" [44].

The authors discuss the usage of Curriculum Learning, where the learning process is structured in a series of increasingly difficult tasks [8]. The approach consists of defining a series of tasks with varying difficulty levels. Initially, the agent is trained on simpler tasks with frequent and clear rewards. These tasks could involve managing traffic at a single intersection or a small network with low traffic volume. As the agent's performance improves, it is progressively introduced to more complex scenarios, such as larger networks with higher traffic volumes and multiple intersections. This gradual increase in difficulty ensures that the agent builds a robust understanding of traffic dynamics and learns effective control strategies.

The methodology also involves setting up the simulation environment, defining the reward structure, and selecting appropriate RL algorithms that can handle the complexities of traffic management. The training begins with the agent interacting with a simulated traffic environment, receiving rewards based on actions. The initial phases of training focus on simple tasks to ensure the agent quickly learns basic traffic management strategies. As the agent's proficiency increases, the tasks' complexity increases. This incremental approach helps the agent transfer knowledge from simpler tasks to more complex ones, enhancing learning efficiency. The training process also involves fine-tuning the hyperparameters of the RL algorithm to optimize performance. Additionally, techniques such as experience replay, where the agent learns from past experiences, and target networks, which stabilize training, are employed to improve learning outcomes.

The research demonstrates that agents trained using this method perform significantly better than those trained using traditional RL approaches. Using curriculum learning leads to faster convergence and higher overall performance in managing traffic flow. The results include quantitative metrics such as reduced waiting times at intersections, lower average travel times, and improved traffic throughput. The study also demonstrates that the agents can generalize their learning to new and unseen traffic scenarios, showcasing the robustness of the curriculum-based approach.

3.4.2 Deep Reinforcement Learning

Deep Learning [39] is a subset of machine learning that focuses on neural networks with many layers to model and understand complex patterns in data. Through the usage of large datasets and high computational power, deep learning algorithms can automatically learn representations from raw inputs. Deep Reinforcement Learning (DRL) [1] combines deep learning with reinforcement learning principles. In DRL, deep neural networks are used to approximate the decision-making policies or value functions that guide agents in an environment. These agents learn to achieve goals by interacting with the environment and receiving feedback in the form of rewards or penalties, creating complex behaviours from raw sensory inputs in the process.

Kheterpal et al. (2018) present "Flow" [36], an open-source framework that leverages deep reinforcement learning to achieve control in traffic simulations. Flow allows researchers to apply RL methods to traffic scenarios, enabling vehicle and infrastructure control in diverse environments. Flow facilitates the development of autonomous vehicles and intelligent infrastructure controllers, optimizing customizable traffic metrics like traffic flow and average velocity.

Flow provides an open-source Python framework that integrates traffic simulators with RL libraries, allowing users to create environments encapsulating MDPs for RL problems. Flow's architecture is designed to be modular and extensible, with components for generating networks and configuring environments. It manages simulation initialization, steps, and resets; defines observation and action spaces; and aggregates information to calculate observations and rewards. Flow supports multiple RL libraries (rllab [24], RLlib [41]) and is compatible with OpenAI's Gym [15]. It also enables the usage of SUMO's microscopic simulation capabilities [38], including various car-following and lane-changing models, to create rich environments for training RL agents.

Flow uses policy gradient algorithms [53] like Trust Region Policy Optimization (TRPO) [65] and Proximal Policy Optimization (PPO) [66] to train controllers, with actions drawn from stochastic distributions to facilitate exploration and gradient computation. Techniques to improve training efficiency include AWS Integration, Parallelization and Policy Evaluation.

3.4.3 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) is a field that extends traditional reinforcement learning to environments with multiple interacting agents. In MARL, agents learn not only from their own experiences but also from their interactions with other agents within the same environment. This collaborative and competitive learning process allows agents to develop strategies that

consider the actions and reactions of others, leading to more sophisticated and realistic models of complex systems.

Lemos et al. (2019) explore the usage of MARL within the context of traffic simulations in their work "Combining adaptation at supply and demand levels in microscopic traffic simulation: a multiagent learning approach" [40]. The authors focus on mitigating traffic congestion by applying Multi-Agent Reinforcement Learning (MARL) by exploring the simultaneous learning of driver and traffic signal agents. They explain how most MARL studies focus on a single class of agents: either traffic lights [45] [37] or drivers and their route choice [58] [7].

The goal of the study is to define a microscopic modelling approach and simulate scenarios where drivers aim to minimize travel costs by selecting efficient routes, while traffic signals strive to maximize traffic throughput at intersections. The study measures travel times from origin to destination and shows that MARL-based approaches can reduce these times, incorporating microscopic simulation that allows for fine-grained modelling of agents, including different speeds, acceleration, car-following behaviour, and traffic signal control.

The MDP for driver agents is based on repeated games, where there is only one state. The actions for drivers are the pre-calculated shortest paths, and the reward signal is the negative of the driver's travel time. On the other hand, the MDP for traffic signals is state-based, considering stochastic games. The state for each traffic signal consists of the current phase, elapsed time of the current phase, and queue lengths for each phase. The actions for traffic signals are either keeping the current phase or changing to another phase. The reward for traffic signals is defined as the difference between the current and previous average queue length.

Driver agents learn by using Q-learning [75] in episodes, where each episode ends when all drivers reach their destinations. Drivers choose actions (routes) based on the Q-values and receive rewards based on their travel time. The Q-values are updated using the Q-learning equation, and the exploration-exploitation trade-off is controlled using an epsilon-greedy exploration strategy. Meanwhile, traffic signal agents also use Q-learning to learn the optimal timing scheme for improving local flow. Traffic signals continuously observe the state of incoming lanes and make decisions to keep or change the green signal. They receive rewards based on the change in queue lengths since the last decision and update their Q-values accordingly.

Mushtaq et al. (2023) explore the integration of MARL in the field of Intelligent Transportation Systems (ITS) in their work "Multi-Agent Reinforcement Learning for Traffic Flow Management of Autonomous Vehicles" [48]. The authors propose an approach that combines Multi-Agent Reinforcement Learning and smart routing to improve the flow of autonomous vehicles in road networks. The study evaluates two MARL techniques, namely Multi-Agent Advantage Actor-Critic (MA2C) and Independent Advantage Actor-Critic (IA2C) [28], along with smart routing for traffic signal optimization. The paper explores non-Markov decision processes to better understand the algorithms. Simulations using the SUMO software modelling tool for traffic simulations are conducted to demonstrate the effectiveness and reliability of the proposed method.

In the IA2C algorithm, each agent learns its own value function and policy from its experiences. The actor loss and critic loss are calculated based on the observed history and the advantage.

The IA2C algorithm allows for rapid exploration by including entropy loss in the policy.

The MA2C algorithm stabilizes the learning process and weak convergence by communicating with neighbouring agents. A spatial discount factor is introduced to weaken the reward and state signals of agents not present in a specific agent's neighbourhood. The learning process becomes more stable, and there is more correlation between local region observations and spatially discounted returns.

Afterwards, a routing strategy is used to manage traffic flow. The Dijkstra algorithm is initially used to compute the shortest paths between origins and destinations. Alternate paths are computed and stored in a route file. During congested hours, vehicles that have not yet reached intersections are considered the current state of the intersection. If the total travel time of the current path and wait time at the intersection is greater than the travel time of the second alternate path, the vehicle is rerouted. This re-routing helps load-balance vehicles and improves traffic flow. The methodology involves the use of detectors, re-routers, and sensors for efficient routing. SUMO provides features for route assignments using re-routers, edge weights, and TraCI functions.

The authors also reference numerous other studies that explore the usage of MARL [18] [6] [29] [17].

3.4.4 Alternatives

There are alternatives to RL for dealing with dynamic environments. In "Engineering Sustainable and Adaptive Systems in Dynamic and Unpredictable Environments" [21], Cardoso et al. (2018) list some socially and biologically inspired techniques suitable for unpredictable contexts:

- **Electronic Institutions (EI):** Electronic Institutions are frameworks designed to facilitate and regulate interactions among autonomous agents within a virtual environment. They provide the rules and protocols that guide agent behaviour, ensuring that interactions are orderly, predictable, and aligned with predefined objectives or social norms. Studies on EI: [56] [54] [55].

- **Evolutionary Computing (EC) and Genetic Programming (GP):** Evolutionary Computing is a subfield of artificial intelligence that involves computational algorithms inspired by the process of natural evolution, such as selection, mutation, and crossover. Genetic Programming is a type of Evolutionary Computing where computer programs are optimized to solve problems by evolving over successive iterations, mimicking the principles of biological evolution. Studies on EC: [20] [19]. Studies on GP: [4] [68] [33].

- **Community Energy Systems (CES):** Community Energy Systems are localized energy networks that generate, distribute, and manage energy within a community. These systems often utilize renewable energy sources and aim to enhance energy efficiency, reduce carbon emissions, and promote energy independence and resilience at the community level. Studies on CES: [35]

3.5 Summary

This chapter explored a variety of studies relevant to the implementation of the proposed work. It examined concepts like Machine Learning (ML), Agent-Based Modeling (ABM), Multi-Agent Systems (MAS), and the integration of these. It focused on traffic simulations and the usage of tools like SUMO and TraCI for modelling traffic systems. The chapter also highlights how these technologies improve traffic flow and decision-making in complex systems, with an emphasis on reinforcement learning.

Chapter 4

Methodology

This section outlines the comprehensive methodology used to address the complexities of flow systems through agent-based modelling and large-scale simulations specifically applied to traffic networks. The methodology is divided into two main parts: the Theoretical Part and the Experimental Setup. The Theoretical Part discusses the conceptual framework, explaining how agent-based modelling and reinforcement learning are employed to optimize vehicle routes and enhance traffic flow. The Experimental Setup details the practical implementation, including the software and hardware used, the preparation process, the input and output data, and the design of traffic network maps and routes. Together, these components provide a robust approach for simulating and analysing traffic systems, aiming to develop efficient routing strategies that can be generalized to other types of flow systems.

4.1 Context of the Methodology

In order to address the intricacies and complexities of flow systems, an approach using agent modelling in large-scale simulations is designed, using traffic networks as a practical application and reinforcement learning for learning and behaviour emergence.

Agent-based modelling allows for the simulation of numerous individual entities (vehicles) within a traffic network, each with its own behaviour and decision-making processes. This scalability is crucial for realistically representing the dynamics of urban traffic systems, where numerous vehicles interact simultaneously. Traffic networks are a practical and tangible application of flow systems, making the research outcomes directly relevant to real-world problems. Urban traffic congestion is a significant issue in many cities, and the ability to develop and test solutions in a simulated environment provides a low-risk, cost-effective way to explore new strategies before implementation. Reinforcement learning provides a framework for agents to learn optimal behaviours through interaction with their environment. This is particularly advantageous in complex systems like traffic networks, where the optimal routing strategy may not be predefined

but needs to be discovered through experience. The ability of reinforcement learning to adapt to changing conditions and continuously improve performance makes it well-suited for dynamic and unpredictable environments. Analysing the behaviours learned by the agents makes it possible to gain valuable insights into the decision-making processes that lead to optimized flow. This is more informative than pre-programmed strategies, revealing the conditions and actions contributing to improved performance. These insights can then be generalized to other types of flow systems, providing a broader applicability of the findings.

The approach consists of simulating a large amount of vehicles performing different trips across traffic networks. Each traffic network follows an architecture representing common traffic design patterns used in real life. The vehicles will have access to a set of functions that allows them to check the system's state and perform different actions. Using reinforcement learning, the vehicles will learn how to use these functions to optimize their trip routes and maximize the system's flow. After they have learned how to do so, it will be possible to inspect what functions were chosen under what conditions at different times of the simulation. The knowledge from this inspection can be extracted and mapped to more general scenarios, thus allowing for the creation of an abstract guidebook on how to ensure flow in any type of flow system.

4.2 Experimental Setup

4.2.1 Software

SUMO was used to simulate the network and the vehicle trips [38]. SUMO, or Simulation of Urban Mobility, stands out as an excellent choice for a vehicle simulator due to its comprehensive features and versatility. SUMO offers a realistic simulation environment, enabling users to model various aspects of urban mobility, including road networks, traffic flow, and vehicle behaviour. Its support for a wide range of traffic scenarios and simulation algorithms makes it suitable for diverse research and development purposes, from studying traffic management strategies to testing autonomous vehicle technologies. Moreover, SUMO's integration capabilities with other software tools enhance its utility and make it a valuable asset for both academic and industrial applications in the fields of transportation engineering and urban planning.

TraCI was used to interact with the SUMO simulations [76]. Integrating TraCI (Traffic Control Interface) with SUMO significantly enhances its capabilities as a vehicle simulator. TraCI enables real-time communication between external applications and the SUMO simulation, allowing for dynamic control and analysis of traffic scenarios. This integration empowers users to implement and evaluate advanced traffic control strategies in a realistic simulation environment, such as adaptive signal control and dynamic route guidance. By leveraging TraCI, developers can interact with SUMO programmatically, accessing detailed information about vehicles, infrastructure, and traffic conditions. This level of interaction facilitates the development and testing of intelligent transportation systems, including connected and autonomous vehicles, as well as traffic

management algorithms. Overall, the utilization of TraCI with SUMO offers unparalleled flexibility and precision in simulating complex urban mobility scenarios, making it an indispensable tool for transportation research and development.

The Ray framework was used for reinforcement learning (rllib) [41]. Ray, renowned for its scalability and efficiency in distributed computing, emerges as an optimal reinforcement learning (RL) framework for integration with SUMO simulations. Ray facilitates seamless incorporation of RL algorithms into SUMO, enabling the training of RL agents to interact with dynamic traffic environments in real-time. Its distributed computing capabilities empower efficient parallelization of RL training, expediting experimentation and policy optimization for traffic control and vehicle management within SUMO's simulation environment. Ray's support for both synchronous and asynchronous RL algorithms caters to diverse training paradigms, making it an ideal choice for researchers and practitioners seeking comprehensive solutions to address various traffic-related challenges. By harnessing Ray's capabilities, namely PPO, alongside SUMO simulations, developers can accelerate innovation in transportation systems, fostering the development of safer, more efficient, and sustainable urban mobility solutions.

The Gymnasium (previously Gym) library was used to connect Ray, TraCI and SUMO [15]. Incorporating the Gymnasium library is crucial for creating a custom environment that connects SUMO, TraCI, and Ray seamlessly. Gymnasium provides a standardized interface for RL environments, streamlining the integration process and enabling efficient communication between these components. By leveraging Gymnasium, developers can effortlessly design custom environments tailored to their specific needs, integrating SUMO's realistic traffic simulations with TraCI's real-time control capabilities and Ray's powerful reinforcement learning algorithms. This interoperability ensures smooth interaction between the simulation, control, and learning components, facilitating the development and evaluation of advanced traffic management strategies and autonomous vehicle systems with ease and precision.

The interaction between all the components can be seen in Figure 4.1. Initially, Python's main function calls for the initialization of the ray framework, which loads the custom environment defined with Gymnasium. Through the custom environment, Ray evokes its reset function, which clears the environment and uses TraCI to start the SUMO simulation. Afterwards, the step function is called repeatedly until it receives a termination variable. In this function, TraCI is used to receive the simulation's state information and to interact with the agents. Once the simulation reaches its end, TraCI closes it and returns a termination variable to Ray through Gymnasium. In Ray's training stage, this process constitutes one episode, and several episodes are executed to constitute one iteration. After each iteration, ray returns its results to Python, and the information is stored. Once all iterations are completed, Ray shuts down, and the program ends.

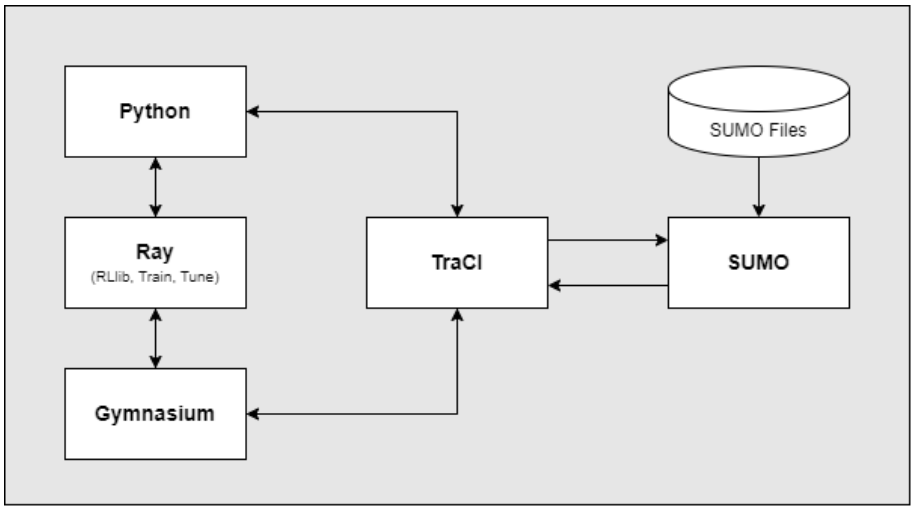

Figure 4.1: Software Component Interactions

4.2.2 Hardware

The computer used to run all the simulations had the following specifications:

- **Processor:** Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
- **Video Card:** GeForce GTX 1050
- **Video Card #2:** Intel(R) UHD Graphics 630
- **Operating System:** Windows 10
- **RAM:** 16 GB

4.2.3 Preparation

In order for the program to inspect the state of the simulation and interact with it, several functions that the program can choose were created. These functions are grouped into different categories and ordered with a specific logic. The created logic was inspired by the educational tool "Scratch" [60]. In Scratch, several premade functionalities take the form of blocks and are organised into different categories. Each category relates to the nature of the functionality (Events, Motion, Control, Sensing, Operation, etc.). Users can select different functionality options and connect them together. The connection of these blocks follows a specific logic in order to achieve a concrete result. Similarly, the logic of the program created consists of the existence of 5 different types of blocks: Decision Point, Agent Selection, Behaviour, Condition Check and Action. These 5 blocks are executed sequentially in the written order. Figure 4.2 displays a visual representation of the sequential order of the execution blocks. The purpose of these types of blocks is the following:

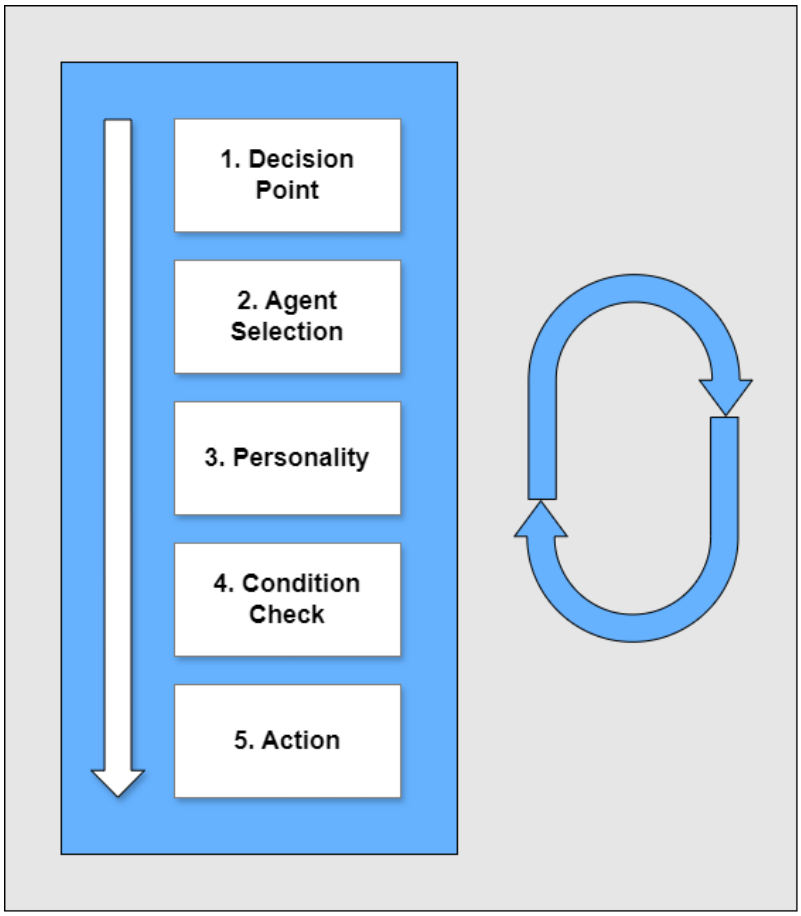

Figure 4.2: Execution Blocks order

1. **Decision Point (DP):** Relates to WHEN the program decides to execute a change in the simulation. Although this five-block combination is created at each timestep, a specific decision point must be selected in order to advance to the following blocks. If a non-empty list of agents is returned by this block, it is considered the selected point for a decision was reached, and it is time for the blocks to be executed.

2. **Agent Selection (AGT):** Relates to WHO should be targeted for this execution. Given that a decision point was reached, it is now decided which agents will be selected for execution.

3. **Personality (PERS):** Related to HOW the selected agents for this timestep will react to the chosen action. This block pretends to mimic the nature of human beings and their likelihood of following suggestions or commands, thus creating a layer where the agent may accept or reject them. Therefore, this block filters the agents that will reject the Condition checked and the Action chosen for them. This block was purposefully placed here to prevent some agents from wasting time with the following blocks in the case of rejecting them by the end,

4. **Condition Check (COND):** Relates to inspecting a variable and checking IF it passes a condition. This block analyses a performance component of the agent itself or the simulation

state and compares it to a given threshold. The result of this comparison is used to decide whether an action should be executed.

5. **Action (ACT):** Relates to WHAT action should be finally executed once all the previous blocks were passed successfully. The action consists of changing how the agent travels through the flow system.

At each timestep in the simulation, the program generates a combination of 5 integers. Each integer works as an index for a map of a possible option for each block type.

Description of the Execution Blocks options:

1. **Decision Point (DP):** WHEN to execute
    - **Check Periodically:** Execute the block sequence every X timesteps.
    - **Check Randomly:** Execute the block sequence with a random chance.
    - **Check Key Points:** Execute the block sequence if agents pass over specific map points in the current timestep.
    - **Check Location-based:** Execute the block sequence if agents are within a selected map region.
2. **Agent Selection (AGT):** WHO to select
    - **Select from All:** Select a subgroup of all agents in the map.
    - **Select from the DP:** Select a subgroup of the agents returned from the Decision Point block.
3. **Personality (PERS):** HOW to behave
    - **Submissive:** High likelihood of an agent accepting the action.
    - **Indecisive:** Medium likelihood of an agent accepting the action.
    - **Stubborn:** Low likelihood of an agent accepting the action.
4. **Condition Check (IF):** IF something checks out
    - **Total Trip Distance:** Checks if the total distance of the agent's trip passes a given condition.
    - **Elapsed Time:** Checks if the agent's trip elapsed time passes a given condition.
    - **Elapsed Distance:** Checks if the distance agent's travelled distance passes a given condition.
    - **Time Left:** Checks if the estimated time left on the agent's trip passes a given condition.
    - **Distance Left:** Checks if the distance left on the agent's trip passes a given condition.

- **Current Speed:** Checks if the agent's current speed passes a given condition.
- **Average Speed:** Checks if the agent's average speed during its trip passes a given condition.
- **Slow Duration:** Checks if the duration that the agent was travelling with a slow speed passes a given condition.
- **Stopped Duration:** Checks if the duration that the agent was stopped passes a given condition.
- **Number of Stop and Start:** Checks if the number of times the agent came to a halt and resumed its travel passes a given condition.
- **Number of Surrounding Agents:** Checks if the number of vehicles surrounding the agent passes a given condition.
- **Traffic in Route:** Checks if the number of vehicles in the remaining route of the agent passes a given condition.

5. **Action (THEN):** THEN execute this action

- **Slow Down:** Decreases the agent's current speed.
- **Reset Speed:** Resets the agent's speed back to normal.
- **Change Lane:** Changes the agent's current lane to the adjacent one, if possible.
- **Avoid Next Edge:** Performs a reroute by avoiding the next edge in the agent's route, if possible.
- **Enter Highway:** Performs a reroute to force the agent to go through a "highway" labelled edge.
- **Enter Avenue:** Performs a reroute to force the agent to go through an "avenue" labelled edge.
- **Go Directly:** Performs a reroute to force the agent to go through the shortest path to its destination

Block Type	Block Option	Value Options				Notes
Decision Point	Check Periodically	2	4	8	16	Return all agents if current_timestep % X == 0
	Check Randomly	0.1	0.2	0.4	0.8	Return all agents if random_number > X
	Check Key Points	0	1	2	3	Return all agents passing over the Key Points on a specific quadrant
	Check Location-based	0	1	2	3	Check [top_left, top_right, bottom_left and bottom_right] quadrants, respectively
Agent Selection	Select from All	0.1	0.2	0.4	0.6	% of agents to select
	Select from the DP	0.2	0.5	0.7	1.0	% of agents to select
Personality	Each personality block option has a single hardcoded value for the agent's likelihood of accepting the action. Submissive: 90%, Indecisive: 50%, Stubborn: 10% Therefore, the personality block does not require value options for its functionality.					
Condition Check	Total Trip Distance	500	750	1000	1250	Check if total_trip_distance > X
	Elapsed Time	20	35	60	80	Check if elapsed_time > X
	Elapsed Distance	100	250	500	1000	Check if elapsed_distance > X
	Time Left	10	20	30	40	Check if time_left > X
	Distance Left	200	350	500	650	Check if distance_left > X
	Current Speed	3	5	8	10	Check if current_speed < X
	Average Speed	5	7	9	11	Check if average_speed < X
	Slow Duration	10	20	30	40	Check if slow_duration > X
	Stopped Duration	5	10	15	20	Check if stopped_duration > X
	Num of Stop and Start	2	4	8	16	Check if num_stop_start > X
	Num of Surrounding Agents	1	2	4	6	Check if num_surrounding_agents > X
	Traffic in Route	0.05	0.1	0.15	0.2	Check if traffic_in_route > X
Action	Slow Down	3	4	5	6	Change the current speed to X (m/s)
	Reset Speed	No value options are required for the functionality of these options.				
	Change Lane					
	Avoid Next Edge					
	Enter Highway					
	Enter Avenue					
	Go Directly					

Table 4.1: Possible Value Options for each Block Option

On top of the integers generated for the block options, 4 other integer variables are generated in each timestep: "dp_val_idx", "agt_val_idx", "cond_val_idx", and "act_val_idx". Each of these integers works as an index for the value to be used in combination with the option chosen for a specific block type. The purpose of these integers is to attribute the model with choice freedom for the values used in some block options. Instead of having a single hardcoded value as a parameter or threshold for a function, the model can choose between a premade list of values, which is different for each block option. It is presented in Table 4.1 all the possible value options for each block option.

For example, if the model chooses the first value option (blue column) for the Condition Check option "Distance Left", the program will check if the agent's distance left in its route is greater than 200 meters. Alternatively, if it chooses the fourth value option (yellow column), it will check if the agent's distance left in its route is greater than 650 meters.

The creation of the 4 value options results from a middle-ground decision between having only one hardcoded value (e.g. always making the program check if an agent's distance left in its route is greater than 400 meters) and having the ability to use any value possible value (e.g. allowing the program to check if an agent's distance left in its route is greater than 1 million meters). By using an approach where the model is limited to four options instead of having the freedom to select a number from an infinite amount, the complexity of the learning curve is greatly reduced while still ensuring some exploration capabilities. The final values for these options result from intuition and much testing. It should be noted how not every block type and block option requires the usage of value options for their functionality.

4.2.4 Input

In Gymnasium, each environment comes with its predefined action and observation spaces, which are described in the environment's documentation. Researchers and developers can easily work with different environments while developing and testing reinforcement learning algorithms.

The Observation Space refers to the set of all possible observations the agent can perceive from the environment. These observations provide information about the current state of the environment and are used by the agent to decide which actions to take. In the case of this project, the following metrics are collected each timestep:

- **"activeNum":** the number of currently active agents in the simulation
- **"emissions":** the amount of emissions currently being expelled by all the agents of the simulation
- **"meanSpeed":** the current average speed of all the active agents in the simulation
- **"occupancy":** the current space occupied by all the active vehicles over the total space occupied by the edges being currently used in the simulation
- **"simTime":** the current timestep in the simulation

The Action Space refers to the set of all possible actions that an agent can take within a given environment. In the case of this project, the actions refer to the index of the block option for each block type to be used in the execution block sequence.

- **"dp":** Index for the Decision Point block option.
- **"dp_val_idx":** Index for the Decision Point value option.
- **"agt":** Index for the Agent Selection block option.
- **"agt_val_idx":** Index for the Agent Selection value option.
- **"pers":** Index for the Personality block option.
- **"cond":** Index for the Condition Check block option.

- **"cond_val_idx":** Index for the Condition Check value option.
- **"act":** Index for the Action block option.
- **"act_val_idx":** Index for the Action value option.

4.2.5 Output

Every time a training iteration returns a new highest average reward value, a checkpoint for the model is created and stored in the device, marking a milestone in its learning process. After the training stage is concluded, the testing stage begins. The highest-scoring checkpoint from the training stage is restored in the testing stage, where one or several final simulations are run. In these simulations, several variables are retrieved from every timestep, allowing for a deeper understanding of the simulation evolution. These variables consist of a history of all recorded observations and actions taken (as described in Section 4.2.4), along with some additional metrics that are not present in the prior: the average trip time, the average trip distance, and a dictionary describing the elapsed time and distance for every vehicle's trip.

This information is then processed and presented to describe the respective simulation of the testing stage, representing the result of the model's learning process. The following output is generated:

- **Observation Graph:** a plot graph displaying, for each timestep, the evolution of "mean_speed", "occupancy", and "activeNum" across the simulation.
- **Action Graph:** a plot graph displaying, for each timestep, the option chosen for each execution block, along with a colour-based representation of the value option chosen for each one. It should be noted that the colour scheme used to represent the value options is the same as the one presented in Table 4.1. The blue dots represent value option 1, the red dots value option 2, the green dots value option 3, and the yellow dots value option 4. The black dots indicate that the respective block option does not require value options.
- **Evaluation Metrics:** a dictionary displaying different metrics for the final values of the simulation:
  - **"mean_speed":** average speed of all vehicles across the entire simulation
  - **"occupancy":** average occupancy of the network edges across the entire simulation
  - **"simTime":** total number of timesteps until the end of the simulation
  - **"avg_trip_time":** average time for every vehicle's trip across the simulation
  - **"avg_trip_distance":** average distance for every vehicle's trip across the simulation

4.2.6 Map and Routes

Exploring simulations across diverse map architectures provides invaluable insights into mobility systems, as these architectures closely resemble real-world scenarios encountered in cities worldwide. By encompassing the most common layouts in urban environments, the simulations cover a wide range of scenarios and offer a comprehensive understanding of transportation dynamics. The chosen architectures for testing are Grid, Radial and Medieval. This diversity ensures that the simulations capture the complexities of real-world traffic scenarios, enabling the development of robust and adaptable solutions across various urban contexts.

- **Grid Architecture:** The orthogonal grid architecture features streets intersecting at right angles, forming a pattern of blocks and squares. This is usually a result of a deliberate, structured urban design.
- **Radial (Spider Web) Architecture:** The decentralized radial city architecture consists of streets extending outward from a central point, connecting to concentrically arranged roads leading to the city's periphery. This is often seen in suburban neighbourhoods.
- **Medieval (Random) Architecture:** Presenting a chaotic and irregular layout, the medieval architecture reflects a blend of strategic defence and organic urban growth. This is commonly present in historic city centres or rural areas.

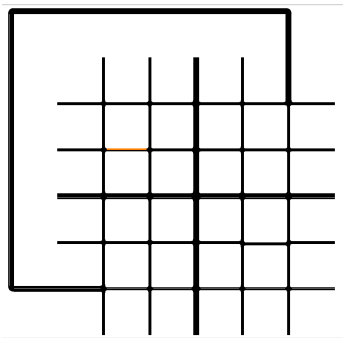

Figure 4.3: Grid Map

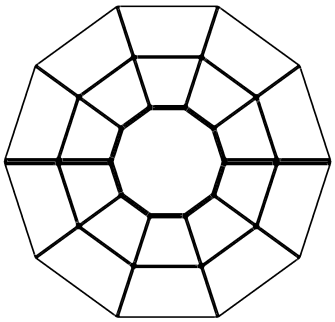

Figure 4.4: Spider Map

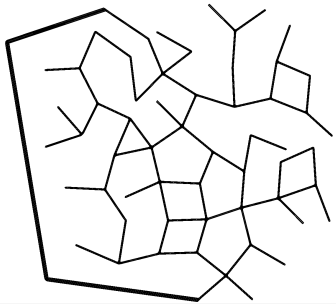

Figure 4.5: Random Map

In Figures 4.3, 4.4 and 4.5, it is possible to inspect the maps created for the Grid, Radial and Medieval architectures, respectively. The maps were generated using the SUMO built-in tools *netgenerate*, which creates abstract road networks from a command line, and *netedit*, which is a visual network editor that allows the modification of aspects of an existing network. Because of the nomenclature used in *netgenerate* to describe the network types, the name of some maps was changed accordingly for simplicity purposes. Because of its similarity to a spider web, the map of the radial architecture takes the name "Spider". Similarly, the map of the Medieval architecture takes the name "Random". The grid architecture remains with the map name "Grid".

The routes for these maps were created with the SUMO built-in tool *randomTrips.py*, where a random set of trips can be configurable and generated. Each set of trips consists of 200 vehicles

that start their travel at different instants with different origin and destination points on a map. After a lot of trial testing, 200 seems to be the appropriate number of trips for creating an environment with enough congestion for the designed maps, while avoiding excessive vehicles in backlog awaiting to enter the simulation. Additionally, 100 different sets of trips were created for each map, ensuring that the model does not overfit while learning how to ensure efficiency for the same set of trips.

4.3 Summary

In summary, this chapter presents a detailed description of the decided methodology used to optimize traffic flow through agent-based modelling and reinforcement learning. The approach utilizes SUMO for realistic traffic simulation and TraCI for dynamic control, while Ray facilitates the integration of reinforcement learning algorithms. The preparation process involved creating functions for decision points, agent selection, behaviour, condition checks, and actions, which are executed in a specific sequence. The experimental setup, including software, hardware, and algorithmic components, is thoroughly explained. The chapter concludes with an explanation of the input and output metrics used to evaluate the effectiveness of the optimization strategy. This methodological framework provides a robust foundation for simulating and improving urban traffic systems.

Chapter 5

Experiments and Results

In this chapter, we present the experiments conducted and the results obtained throughout the project. The experiments were designed to iteratively improve the reward function and the overall performance of the agents in the simulation environment. Various configurations and modifications were tested to address the initial issues observed with the baseline performance and to enhance the model's effectiveness. The results of these experiments provide valuable insights into the effectiveness of the implemented strategies and the improvements achieved over the control group.

5.1 Experiments

A Control Group (CG) was conceived to create a baseline for comparing and analysing the results of the experiments. The Control Group consists of the evaluation scores for a simulation being executed without any intervention by the agents on every map. This project aims to achieve evaluation scores that are better than the CG for each map.

In an earlier project stage, the reward function consisted of the reward sum of 6 variables: "arrivedNum", "collidingNum", "simTime", "fuel", "emissions", "meanSpeed", and "occupancy". Each variable's expected values during the simulation were analysed at different timesteps. Afterwards, a spreadsheet table describing the score Y for each possible value of the observation X for each variable was written. These scores are greater the better the result is than its expected values, and the opposite is true for worse results, even going below zero if considered bad enough. This way, the desired results are rewarded positively, and the undesired ones are penalised. Then, by using polynomial regression, a mathematical function that accurately represents the scoring for each possible value was generated for each of the 5 variables. During every timestep, a score was associated with each variable according to their respective polynomial function, and the reward resulted in the sum of these scores.

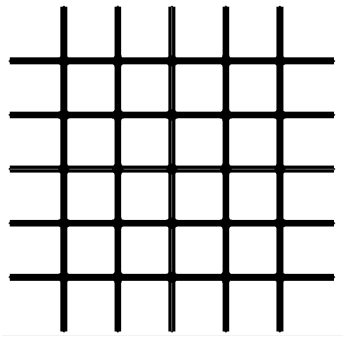

Figure 5.1: Previous Grid Map

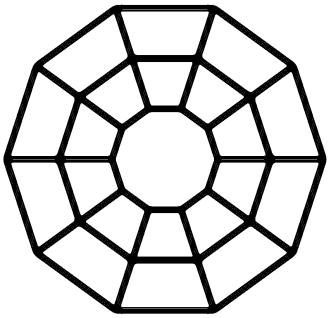

Figure 5.2: Previous Spider Map

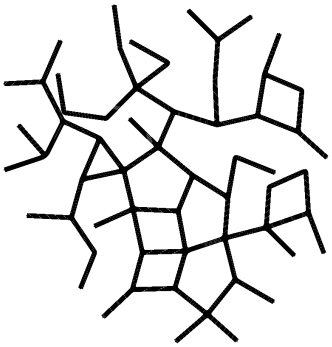

Figure 5.3: Previous Random Map

Although apparently promising, results from this setup often returned an evaluation score lower than the control group. Additionally, many results scored the same as the control group, because the model learned to create execution block sequences that lead to the absence of action executions, which indirectly led to the non-intervention of the agents and, therefore, the same case as the control group. For example, the model would learn to always pick the "stubborn" personality option or the "resetSpeed" action option, which would essentially lead to inactivity in the simulation. To fix this, many changes were made iteratively.

Firstly, some variables were removed since they were redundant, irrelevant, or simply linearly dependent on other variables, which unnecessarily increased the complexity of the reward function and, thus, negatively affected the learning process. This was the case for "arrivedNum" and "collidingNum", "fuel", "emissions" and "occupancy". This way, only the "meanSpeed" and "simTime" variables were considered for the reward function, since they are the only essential ones for representing the flow efficiency of a system.

Secondly, the polynomial regression seemed promising as a nuanced way of mapping from observations to rewards and its capability of modelling complex relationships. However, it ultimately led to very sharp gradients in the reward landscape that were difficult for the RL model to learn from. Because of this, the reward was greatly simplified and started using roughly the direct values of the considered variables instead of a scoring map to possible values.

Thirdly, penalties were added for the choice of execution block options that could lead to the absence of any execution. This way, the model won't get stuck in a local optima where it does not intervene in the simulation and will explore execution options that return an evaluation score higher than the Control Group.

In spite of all of these improvements in the logic of the program, the problem of not achieving better results than the control group persisted. There were some rare instances where the simulations returned better evaluation metrics, but the results were unsatisfactory. In fact, this happened when an action option called "SpeedUp" existed, and the model was learning to simply increase every vehicle's speed (beyond the speed limit) for every timestep. Although this made sense in theory, in practice, these were not viable nor innovative results.

At this point, it was understood that the problem was not on the program's logic, but most

likely on the networks designed. At first, the maps used were not the ones displayed in Section 4.2.6 of page 39. Initially, motivated by the idea of Curriculum Learning, the maps were much simpler (Figures 5.1, 5.2 and 5.3), with intentions of making them more complex as the model made progress in the learning process. However, it turned out that the maps were too simple and homogeneous for any progress to have been achieved. For each type of network, every edge had two lanes, the same speed limit, standard intersection priority, and bidirectional connections. In hindsight, it makes sense that if there are no different aspects of the network for the agents to exploit, then every route option will be considered equally good. In this environment, there is no possibility of improvement in traffic flow other than just increasing the vehicle's speeds. In order to tackle this issue, several changes were made to these maps, with the aim of making them more heterogeneous:

- **Traffic lights:** priority control in some of the intersections.
- **Change in the number of lanes:** edges with one, two or three lanes in different network sections.
- **One-way streets:** sections where traffic only flows in one direction.
- **Blocked lanes:** lanes that are inaccessible, to simulate traffic accidents or congestions.
- **Avenues:** edges with priority over others and/or with higher speed limits.
- **Highways:** edges with more lanes and significantly higher speed limits surrounding the network.

The purpose was to add some traffic elements to increase the heterogeneity of the environment while still trying not to make it too context-dependent. Upon implementation, there were ideas of adding elements like parking areas, pedestrian crosswalks, and bus stops, but it was decided not to. This decision stemmed from fear that the environment would become too closely related to its practical application to generate suitable solutions for abstract scenarios.

The modification of these maps alone was not sufficient for immediate improvements in the simulation results. It was at this stage that the actions "enterHighway", "enterAvenue", and "goDirectly" were implemented, so as to make the best use of these newly added components. Additionally, value options for the decision point option "checkKeyPoints" were added, in order to provide the model with better specificity for when to execute and which agents to select. Lastly, removing traffic lights from every network and not using a different set of trips throughout the training simulation finally led to good results. This is not surprising since the traffic lights followed a static program, and improving flow through improvement of the traffic light system is a whole field of study on its own, as seen in the State-of-the-Art chapter. Since this project focuses on improving the route of the vehicles and not on traffic lights, the latter was completely removed.

5.2 Results

Although "mean_speed", "simTime", and "occupancy" are the selected evaluation metrics, "simTime" was considered the most important one when deciding if a result is better than the control group. Because the number of total vehicles is always the same, the faster it is for all vehicles to arrive at their destination, the higher the vehicle throughput of a network will be, which ultimately represents a higher flow in the system. This section presents the best scoring results for each of the designed networks, along with an interpretation of them.

5.2.1 Grid

The control group score for the 'Grid' network is:

- **'overall_mean_speed':** 9.361
- **'avg_occupancy':** 0.126
- **'sim_time':** 229
- **'avg_trip_time':** 96.259
- **'avg_trip_distance':** 611.538

The best scoring result achieved for the 'Grid' network is:

- **'overall_mean_speed':** 9.4
- **'avg_occupancy':** 0.125
- **'sim_time':** 209
- **'avg_trip_time':** 95.259562
- **'avg_trip_distance':** 618.279

The observation and action graphs for the "Grid" network can be seen in Figures 5.4 and 5.5, respectively. After analysing the latter, it is clear there are some notable trends in the choice of the execution block options.

Overall, as a first impression, it is noticeable that there is a lack of preference for the value option 3 (represented by the green dots), meaning the model learned to favour both extreme ends of the value options: really low values (blue) and really high values (yellow).

Firstly, under the Decision Point block, there is a clear preference for a random check with low probability, along with a check on agents passing by key points in the network. This reveals that the model chose to execute mostly when there are agents passing over these key points, alternating with a low probability of executing randomly, opting for inactivity in these cases.

Secondly, under the Agent Selection block, there is no strong preference for any block option. Agents are selected from the result of the previous block and from the list of all agents in the

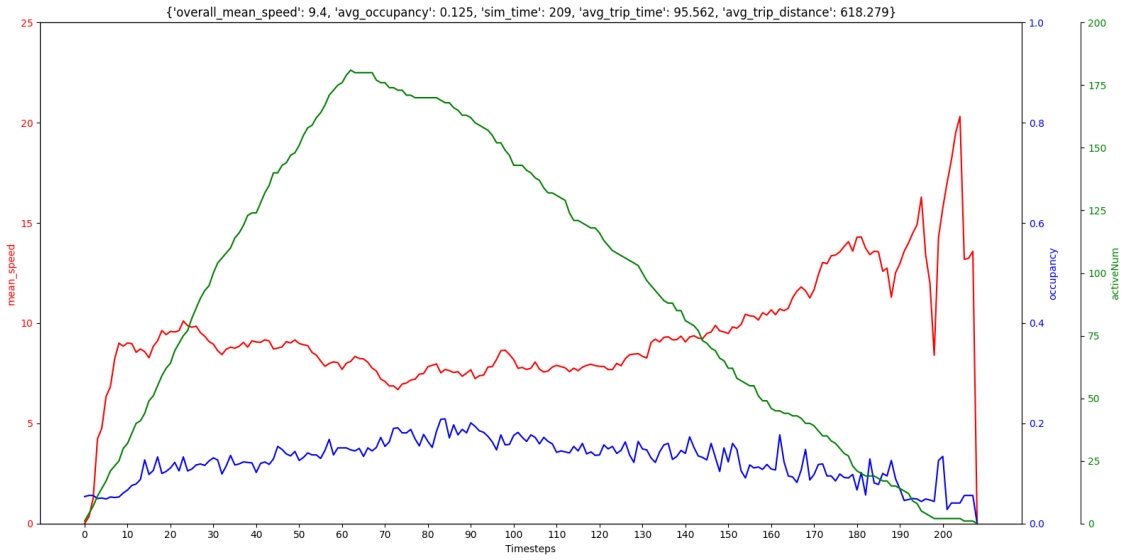

Figure 5.4: Grid: Best Result Observation Graph. The graph displays the state of the system for every timestep of the simulation: the average speed of all agents (red), the average occupancy of the network edges (blue), and the number of active numbers. It shows a steady and natural progression of these metrics.

simulation equally. Regardless, there is a strong preference for the value option 1 (blue), which means the model mostly selects a small number of agents for the following blocks.

Thirdly, under the Personality block, there is a strong preference for "indecisive" behaviour, meaning half of the time, the selected agents will accept a change in their behaviour.

Afterwards, the model learned that the most important Condition Check options to analyse are the agent's speed and the number of surrounding agents, equally choosing the extreme option values (blue and yellow). Regarding the agent's speed option, the yellow dots represent higher sensitivity, since the condition check will pass the comparison if the agent has any speed below 11 m/s. Alternatively, the blue dots represent higher sensitivity in the option regarding the number of surrounding agents, since the comparison will pass if there is more than one agent surrounding the agent. The opposite happens when the blue and yellow options are chosen, respectively.

Lastly, there is a clear preference for the "goDirectly", "avoidNextEdge", and "resetSpeed" options in the Action block. The lack of preference for "enterAvenue" means the model avoids clogging the centre edges of the network. This makes sense in a grid-like network because the agents often enter from the fringe and need to traverse the centre to reach their destination. Interestingly, the model finds it unnecessary to use the surrounding highway in order to achieve this. Apparently, the model finds success in combining "avoidNextEdge" with "goDirectly", which results in a small detour in the agent's route. The option "resetSpeed" represents an absence of change in the agent's behaviours, since the "slowDown" option is practically never chosen.

Ultimately, it seems the model learned to combine the block options and value options to rarely execute, and retrieve a small amount of agents when it does. Upon execution, this small set of agents considers its speed and the number of surrounding agents and executes a small detour

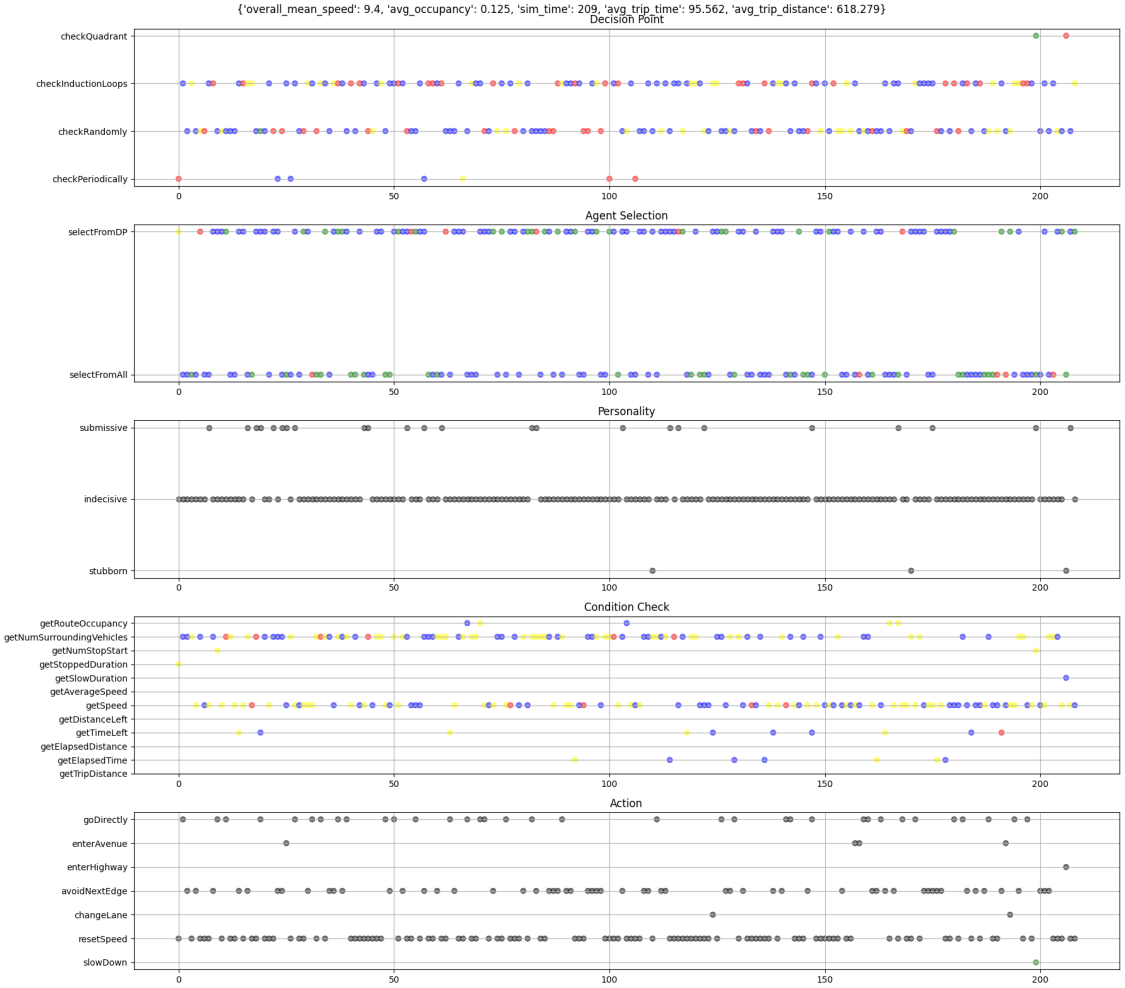

Figure 5.5: Grid: Best Result Action Graph. The graph displays, for each timestep, the option chosen for each block type (Decision Point, Agent Selection, Personality, Condition Check, and Action). The coloured dots also represent the value option associated with each block option choice. It shows a contingent block option choice and value option choice.

in order to reduce congestion in the centre of the network. This strategy leads to a slight global improvement in the system's overall speed and average trip time, at a cost of a higher trip distance. Nonetheless, this led to a shorter simulation time for all agents to leave the network, which represents a better system flow.

5.2.2 Spider

The control group score for the "Spider" network is:

- **'overall_mean_speed':** 9.215
- **'avg_occupancy':** 0.127
- **'sim_time':** 235

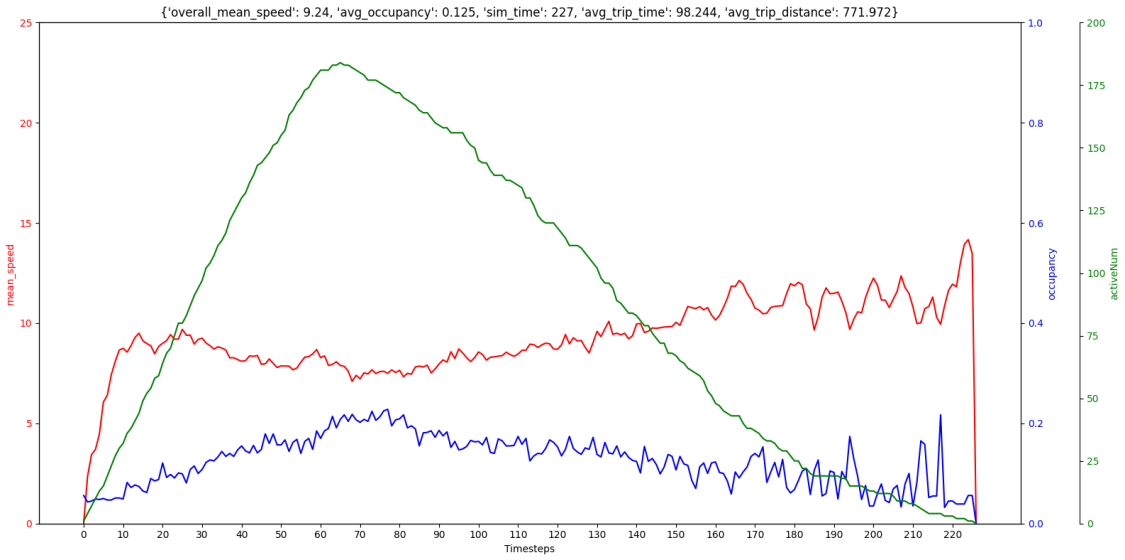

Figure 5.6: Spider: Best Result Observation Graph.The graph displays the state of the system for every timestep of the simulation: the average speed of all agents (red), the average occupancy of the network edges (blue), and the number of active numbers. It shows a steady and natural progression of these metrics.

- **'avg_trip_time':** 98.06
- **'avg_trip_distance':** 748.354

The best scoring result achieved for the 'Spider' network is:

- **'overall_mean_speed':** 9.24
- **'avg_occupancy':** 0.125
- **'sim_time':** 227
- **'avg_trip_time':** 98.244
- **'avg_trip_distance':** 771.972

The observation and action graphs for the "Spider" network can be seen in Figures 5.6 and 5.7, respectively.

Overall, at first glance, it is clear the choice of block and value options is more scattered than the "Grid" result. Although there are some notable preferences, it seems every block option is used at some point in the simulation, even if just once.

Firstly, regarding the Decision Point options, there is a stronger preference for the quadrant check, followed by periodic check and key point check, with the random check coming in last place. This instantly reveals that, in contrast to the "Grid" result, the model learned to initiate the execution blocks more often, not opting as often for options that may lead to inactivity.

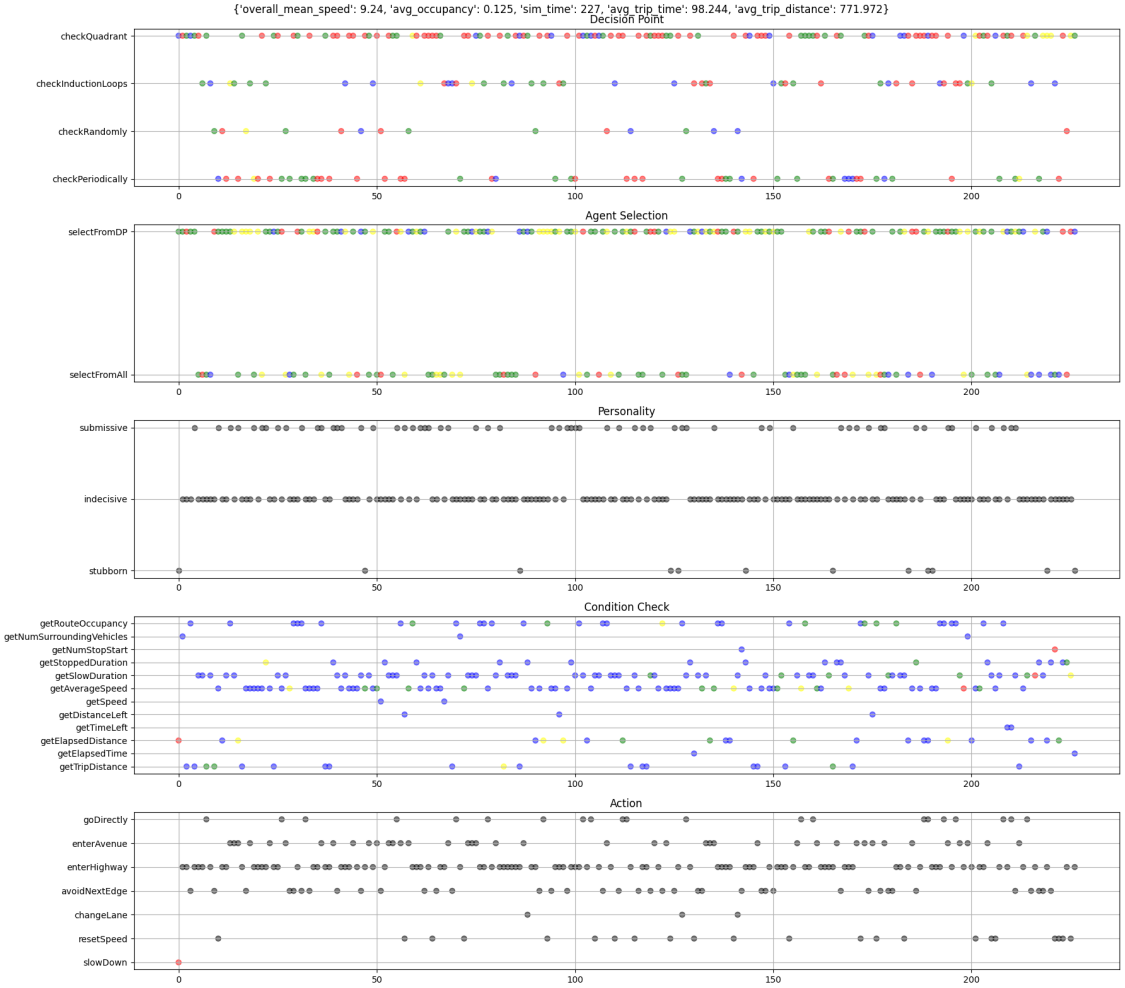

Figure 5.7: Spider: Best Result Action Graph. The graph displays, for each timestep, the option chosen for each block type (Decision Point, Agent Selection, Personality, Condition Check, and Action). The coloured dots also represent the value option associated with each block option choice. It shows a scattered block option choice and a contingent value option choice.

Secondly, in the Agent Selection, there is again an equal preference for both options. However, compared to the "Grid" result, the model learned to choose value options that select a larger set of agents. Likewise, there is a strong preference for value options 3 (green) and 4 (yellow).

Thirdly, the model has a clear preference for the "submissive" and "indecisive" Personality options, further confirming the need for the agents to accept behavioural change.

Afterwards, in the Condition Check block, the model shows a higher preference for a value option rather than any block option. The favouritism for the value option 1 (blue) reveals that the condition check comparisons are easy to pass. There is also some favouritism for the "getSlow-Duration" and "getAverageSpeed" block options.

Lastly, the model considerably prefers the Action block option "enterHighway". In this network, the "highway" labelled edges refer not to a highway but rather to the outermost ring of the map. Although these edges have only one lane and cover the biggest distance, the model still

learned to make the agents take this path over the centre-most ring. Additionally, there is some preference for the "enterAvenue" and "avoidNextEdge" action options.

Ultimately, the model has learned to execute most of the time and retrieve the highest amount of agents. This can be seen in the preference for targeting the agents in entire quadrants, keeping most of them through a selection of a large partition, ensuring they will accept a change in behaviour, and that the condition checks are easily passable. By the end, there is a strong interest in these agents travelling through the longest ring, which is revealed in the significant increase in the average trip distance compared to the control group. And even though the overall mean speed and average trip time remained basically the same, the time for all the agents to complete their trips decreased (shorter simulation time), which means an increase in the system's flow. This means that the model focused on removing agents from the centre-most ring in order to avoid congestion. Nonetheless, actions like "enterAvenue" and "avoidNextEdge" also reveal themselves as crucial to keeping the balance in the distribution of the agents across the network, since only choosing "enterHighway" would most likely lead to even worse congestion problems.

5.2.3 Random

The control group score for the 'Random' network is:

- **'overall_mean_speed':** 11.754
- **'avg_occupancy':** 0.144
- **'sim_time':** 464
- **'avg_trip_time':** 178.154
- **'avg_trip_distance':** 959.32

The best scoring result achieved for the 'Random' network is:

- **'overall_mean_speed':** 10.961
- **'avg_occupancy':** 0.149
- **'sim_time':** 439
- **'avg_trip_time':** 178.254
- **'avg_trip_distance':** 1050.631

The observation and action graphs for the "Random" network can be seen in Figures 5.8 and 5.9, respectively.

Overall, at first glance, it is noticeable how the block option choice is contingent similar to the "Grid" results, and the value option choice is scattered similar to the "Spider" result.

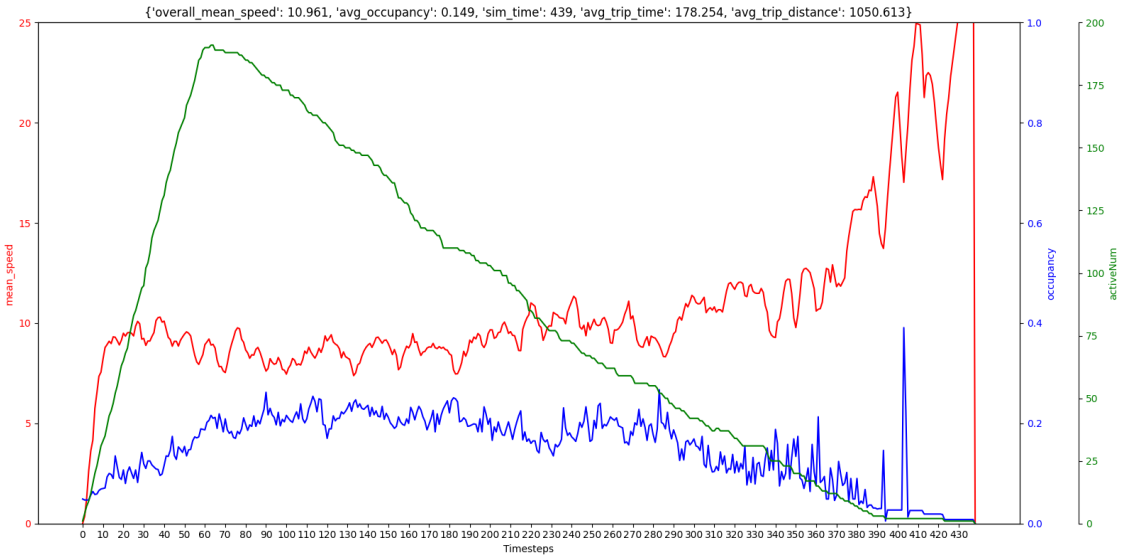

Figure 5.8: Random: Best Result Action Graph. The graph displays the state of the system for every timestep of the simulation: the average speed of all agents (red), the average occupancy of the network edges (blue), and the number of active numbers. It shows a slow progression of the metrics, with high peaks at the end.

Firstly, in the Decision Point section, the model has a strong preference for the "checkQuadrant" option, followed by a preference for the "checkRandomly" option. In both cases, the choice for the value option does not seem to follow any pattern, which means this section targets a diverse range of agents.

Secondly, as seen in the previous results, there is no strong preference for any Agent Selection block option. Additionally, it seems there is also no pattern in the choice for the value option, thus selecting a diverse range of agents for the next blocks once more.

Thirdly, there is a strong preference for the "submissive" and "indecisive" Personality block options, just like in the "Spider" result. In this case, the "submissive" option is chosen even more often than in the "Spider" result, which highlights a strong incentive for the agents to accept a change in their behaviour.

Afterwards, it seems there is a preference for the Condition Check options "getNumStopStart", "getAverageSpeed", and "getSpeed", in descending order of preference. In spite of this, it appears the choice of value options is quite scattered, which leads to a wide variety of value comparisons for the condition checks.

Lastly, the model has learned to mostly favour the "goDirectly" and "enterAvenue" Action block options. In this scenario, the "avenue" labelled edges have higher priority and speed limit than the others, serving a similar purpose as a road connecting distant small towns in a rural district. This means that the model highly incentivizes the agents to enter the avenue and then travel to their destination from there. There is also some preference for the "avoidNextEdge", which means there is some incentive for the selected agents to take a small detour from their current route, allowing other agents to enter the so-favoured avenues and reduce the overall congestion.

Figure 5.9: Random: Best Result Action Graph. The graph displays, for each timestep, the option chosen for each block type (Decision Point, Agent Selection, Personality, Condition Check, and Action). The coloured dots also represent the value option associated with each block option choice. It shows a somewhat contingent block option choice and scattered value option choice.

Ultimately, although there is a concrete trend for the block option choice, the value option choice seems to be too random. The model has learned to essentially retrieve the agents from the quadrants, make an apparently random selection of them, and ensure they are receptive to changes in their behaviour. Afterwards, the model mostly checks their current speed, average speed, or the number of times they stopped and started their travel, and compares it with an apparently random value. Finally, the agents are incentivized to either keep on the shortest path to their destiny or to use the avenue that crosses the network. It is interesting to notice how, in spite of a decrease in the overall mean speed, a significant increase in the average trip distance, and an unchanged average trip time, the number of total timesteps in the simulation shortened significantly. It seems that the model does not find it relevant to rely on the value option choices and that pushing a varying amount of agents to travel through the faster edges seems to be the simplest and easiest option to increase the system's flow.

5.3 Summary

This chapter detailed the extensive experimentation process taken to refine the reward function and improve agent performance in a simulation environment. Initially, the control group provided a baseline for comparison, revealing that early reward function setups led to evaluation scores often lower than or equivalent to the control group. Several iterations and modifications, including variable reduction and reward function simplification, were implemented to address these issues. The introduction of traffic elements and the adaptation of maps increased heterogeneity and provided better learning opportunities for the agents. Despite these enhancements, achieving results consistently better than the control group remained challenging. Ultimately, the experiments highlighted the importance of environment complexity and reward structure in training effective models, leading to notable improvements in specific scenarios. The results for the network configurations 'Grid', 'Spider', and 'Random', showcased the model's learning progress and the strategies it adopted to enhance traffic flow and system efficiency. Table 5.1 shows an overview of the strategies learned by the model for each of the network configurations.

	Grid	Spider	Random
Decision Point	Execute rarely and retrieve agents passing over specific points	Execute almost always and retrieve agents by zones or specific points	Execute often and retrieve of agents by zones
Agent Selection	Select a small set of agents	Select a large set of agents	Select a random set of agents
Personality	Indecisive	Indecisive or submissive	Indecisive or submissive
Condition Check	Inspect the agent's speed and number of surrounding agents; Create extreme passing conditions	Inspect (mostly) the agent's average speed and slow duration; Create easy passing conditions	Inspect (mostly) the agent's number of stop & start, speed and average speed; Create random passing conditions
Action	Do nothing or avoid the next edge to perform a quick detour and go directly to the destination	Mostly detour through the outermost ring. Sometimes detour through the avenue or avoid the next edge	Mostly detour through the avenue and go directly to the destination; Sometimes avoid the next edge

Table 5.1: Overview of the model's learned strategies for increasing the flow in network configuration: Grid, Spider and Random

Chapter 6

Conclusions and Future Work

This chapter concludes the dissertation by summarizing the key findings and accomplishments achieved through the practical application of a traffic network to analyse flow systems. It highlights the creation of a comprehensive framework for large-scale traffic simulations, focusing on agent-based modelling and deep reinforcement learning. The chapter also outlines the limitations and assumptions of the study and suggests directions for future work to further enhance the research.

6.1 Conclusions

In conclusion, this dissertation successfully used the practical application of a traffic network to analyse the intricacies and complexities of flow systems. A framework designed for creating, observing, and evaluating large-scale traffic simulations was constructed, focusing on sophisticated modelling of the agents. Additionally, this framework uses specialized machine-learning technologies (deep reinforcement learning) that allow the agents to learn how to improve the flow of the system under different network scenarios. The agents have at their disposal an easily configurable and extensible set of functionalities that follow a logical sequence of operations for inspecting and interacting with the environment, which can easily be abstracted to any type of flow system. After conducting several experiments with different network schemes, an analysis of the best scoring results was made to understand which is the best way of achieving better flow conditions in a system. Table 5.1 of page 52 displays the sequence of operations that improved the flow for each type of network. However, this study did not manage to develop a scripting declarative language to define sophisticated agent behaviours to apply in generic flow systems.

The work presented was done under some assumptions and presents some limitations. The created framework assumes that a central entity has information regarding numerous aspects of the simulation, like the state of each agent and each network component. Although the experiments returned positive results for the considered situations, they did not explore testing scenarios with variance on certain variables. For example, testing with a different number of agents, different

network architectures, and different map sizes, just to name a few. Additionally, an approach where every agent could individually choose a pair of Condition Check and Action options was attempted, but was left unexplored due to hardware limitations.

6.2 Future Work

Since the presented work did not manage to achieve all the initially proposed goals, the biggest opportunity for future work on this study must be the development of the scripting declarative language for abstract applications.

Additionally, on top of exploring all the previously listed limitations, it would be beneficial to fine-tune the results by extending the learning phases and conducting hyperparameter tuning.

Lastly, it would be very convenient for researchers if a GUI capable of setting up the environment of a flow simulation was created. This way, researchers could intuitively conduct experiments and explore the effects different conditions and execution block options have on flow systems. Ideally, it would be possible to pick and drag boxes relating to the different execution blocks, similar to Scratch.

Appendix A

Algorithms

A.1 Environment Reset and Simulation (Re)Start

This function is evoked by Ray through Gymnasium each time the custom environment needs to be reset to start or restart a simulation.

Algorithm 1 reset()

```
1: if simulationIsRunning() then
2:     closeSimulation()
3: end if
4: resetSimulationVariables()
5: startSimulation()
```

A.2 Simulation Step Loop

This loop represents the step function evoked by Ray through Gymnasium every timestep. The variable **initialize** works as a global variable, stored in the custom environment class.

Algorithm 2 step()

```
1: initialize done ← False
2: while not done do
3:     action ← generateAction()
4:     executeBlock(action)
5:     advanceSimulation()
6:     obs ← getObservation()
7:     reward ← getReward(obs, action)
8:     if simulationReachedEnd(obs) then
9:         done ← True
10:    end if
11:    saveInfo(obs, reward)
12: end while
13: closeSimulation()
```

A.3 Observation collection function

Relates to the **getObservation**() function called in the step() function.

Algorithm 3 getObservation()

```
observed_edges ← [ ]
all_obs ← [ ]
all_vehicles ← getActiveVehicles()
for vehicle in all_vehicles do
    updateVehicleSpeeds(vehicle)
    updateVehicleQuadrantLocation(vehicle)
    updateElapsedTripTimesAndDistances(vehicle)
    if not hasObservedAllEdges(observed_edges) then
        edge ← getCurrentEdge(vehicle)
        obs ← getEdgeObservations(edge)
        all_obs.insert(obs)
        observed_edges.insert(edge)
    end if
end for
obs_metrics ← unpackObs(all_obs)
return obs_metrics
```

A.4 Reward calculation function

Relates to the **getReward**(obs, action) function called in the step() function.

Algorithm 4 getReward()

```
current_timestep, mean_speed ← inspectObservation(obs)
timestep_threshold ← getTimestepThreshold()
if current_timestep > timestep_threshold then
    reward ← -1
else
    reward ← mean_speed
    reward ← normaliseReward(reward)
    penalty ← calculatePenalty(action)
    reward ← reward - penalty
end if
return reward
```

Appendix B

Additional Results

B.1 First Observation Graphs

B.1.1 Linear Scale

Observation graph with previous observation variables, in linear scale. At this implementation stage, the reward was dependent on the number of arrived vehicles.

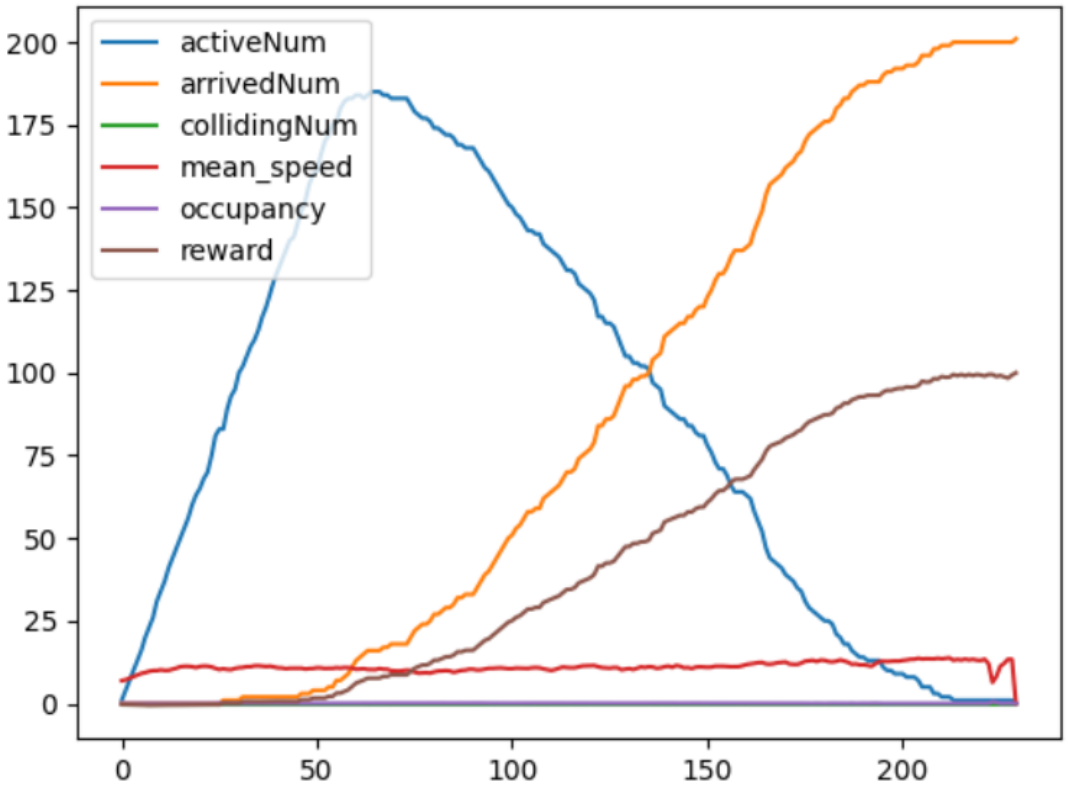

Figure B.1: Observation Graph: Linear Scale

B.1.2 Logarithmic Scale

Observation graph with previous observation variables, in logarithmic scale. At this implementation stage, the reward was dependent on the number of arrived vehicles.

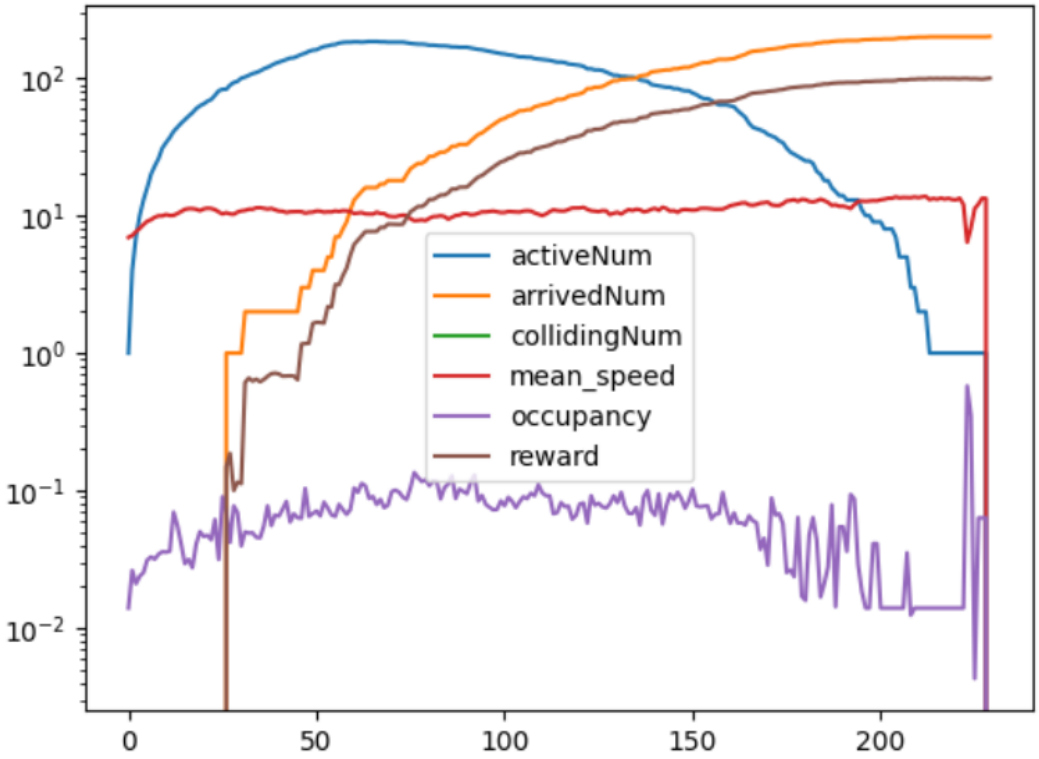

Figure B.2: Observation Graph: Logarithmic Scale

B.1.3 Fuel and Emissions

Observation graph displaying "fuel" consumption and particle "emission" variables. This highlights how the variables are linear dependent on one another, which is why they were simplified in a later stage.

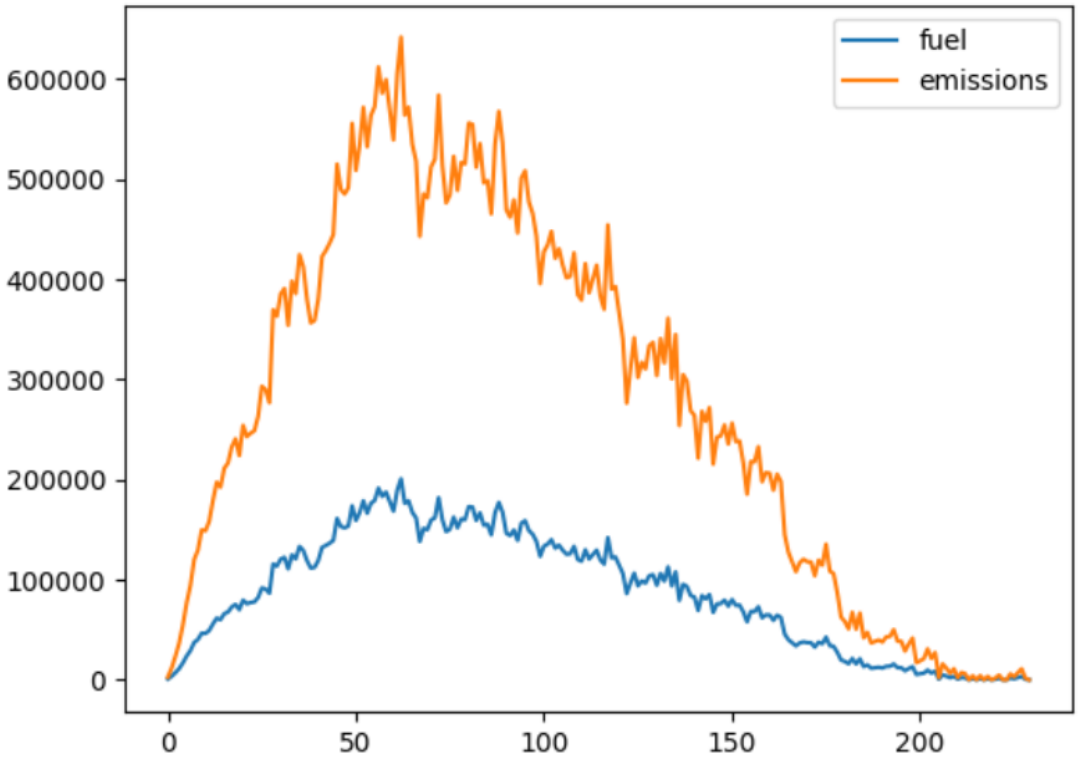

Figure B.3: Observation Graph: Fuel and Emissions

B.1.4 Reward Function from Polynomial Regression

Observation graph displaying the first attempt at designing the reward function with polynomial regression functions for each observation variable.

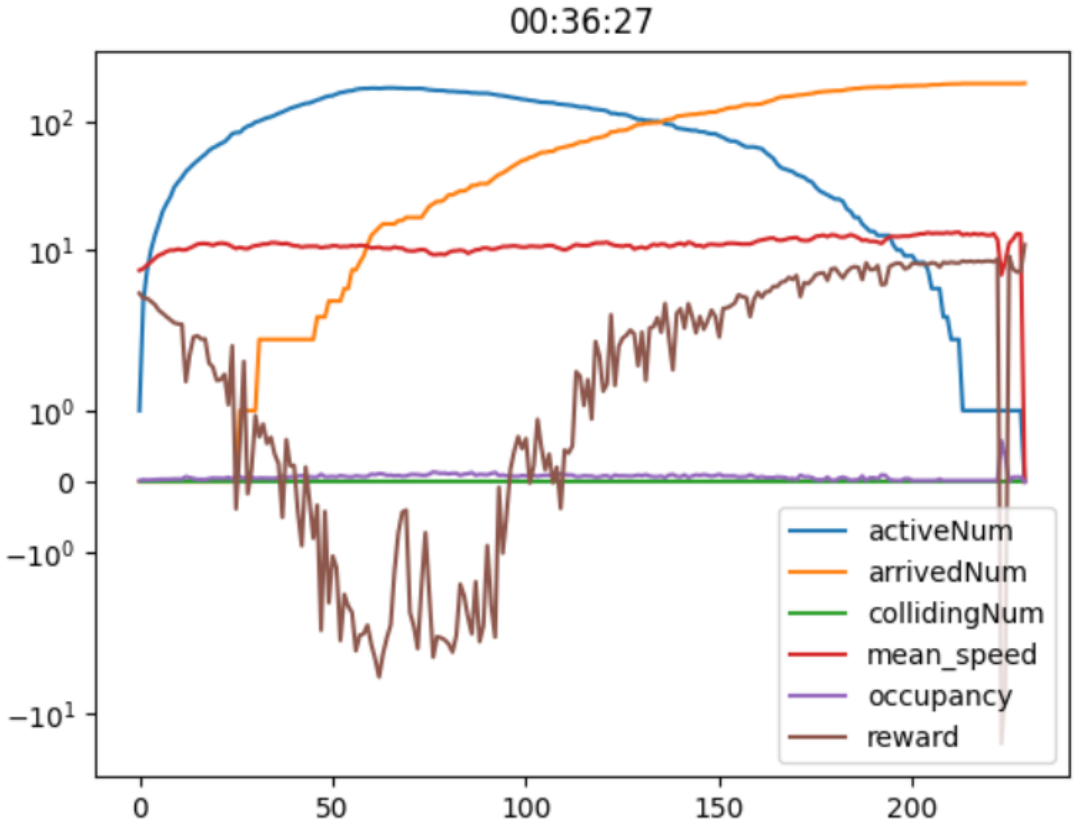

Figure B.4: Observation Graph: Reward Function from Polynomial Regression

B.1.5 Reward Function from Polynomial Regression 2.0

Observation graph displaying the another attempt at designing the reward function with polynomial regression functions for each observation variable, highlighting the respective function for each one.

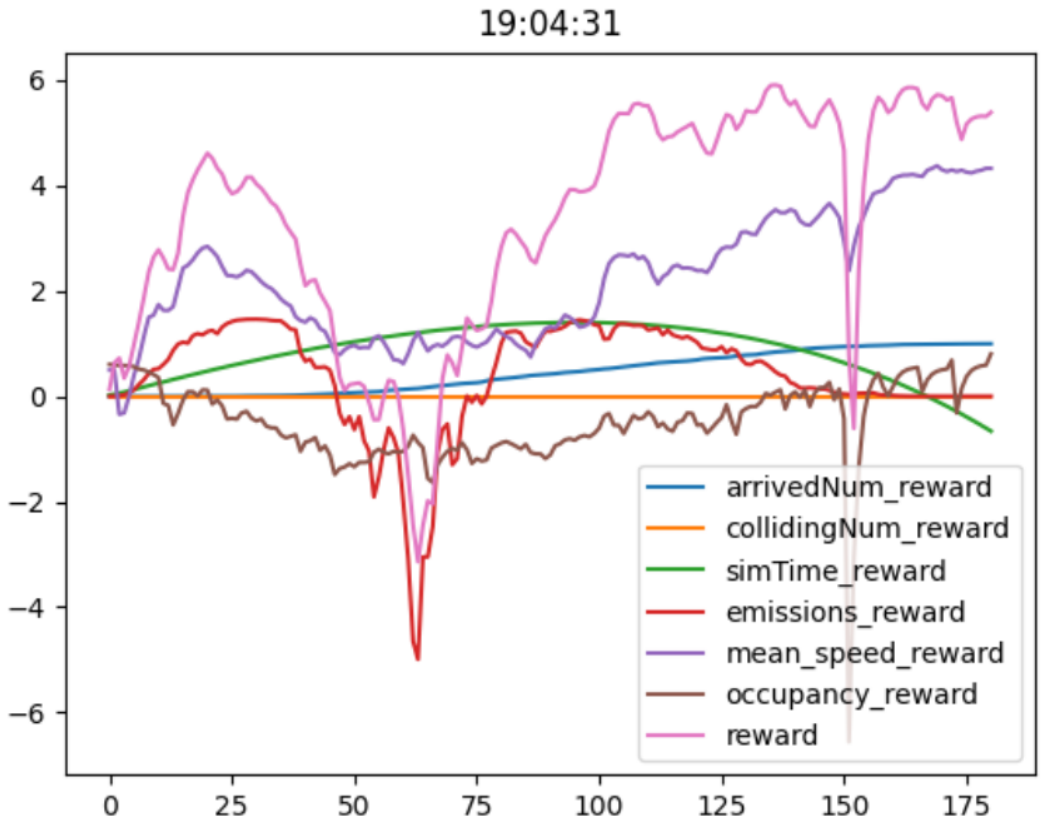

Figure B.5: Observation Graph: Reward Function from Polynomial Regression 2.0

B.2 First Action Graphs

B.2.1 With Execution Block

Action graph displaying an implementation stage when the block type "Execute" still existed. The block contained the options "Execute" and "Standby", which served as a step before the Decision Point where the program would decide if it would execute or not. As displayed by the graph, the program revealed issues where the "Standby" option would be chosen consistently, leading to inactivity from the program on the simulation.

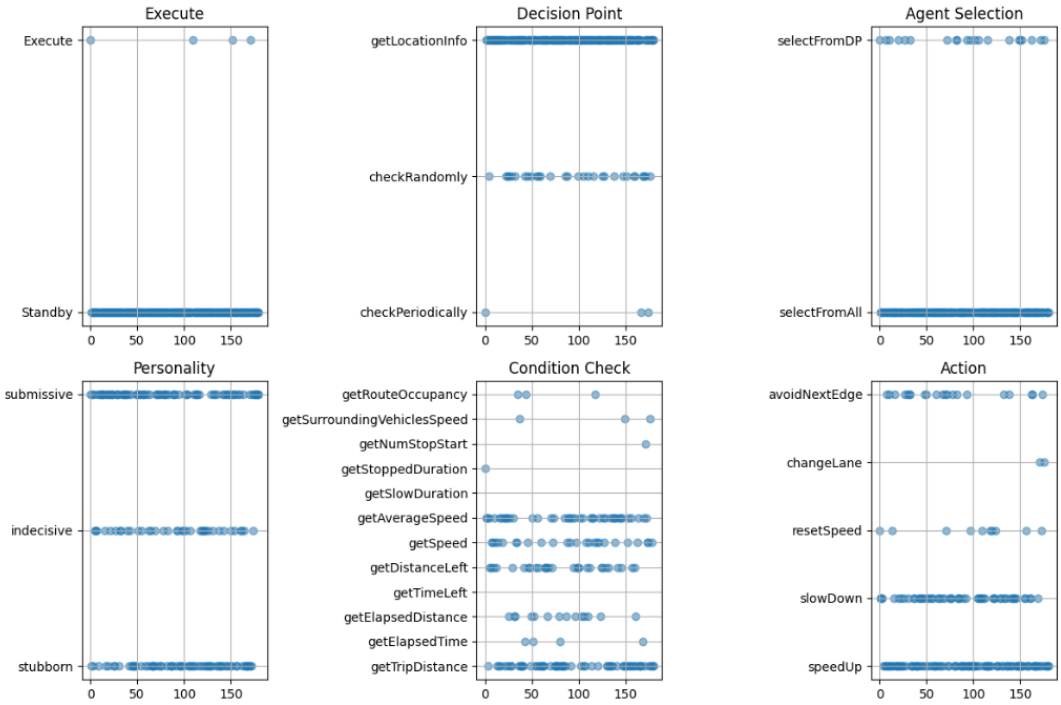

Figure B.6: Action Graph: With Execution Block

B.2.2 Without Execution Block

Action graph displaying an implementation stage when the "Execute" got removed. As displayed by the graph, the model learned to choose other options in different blocks that led to the same problem. In this case, the program learned to always choose the "stubborn" option.

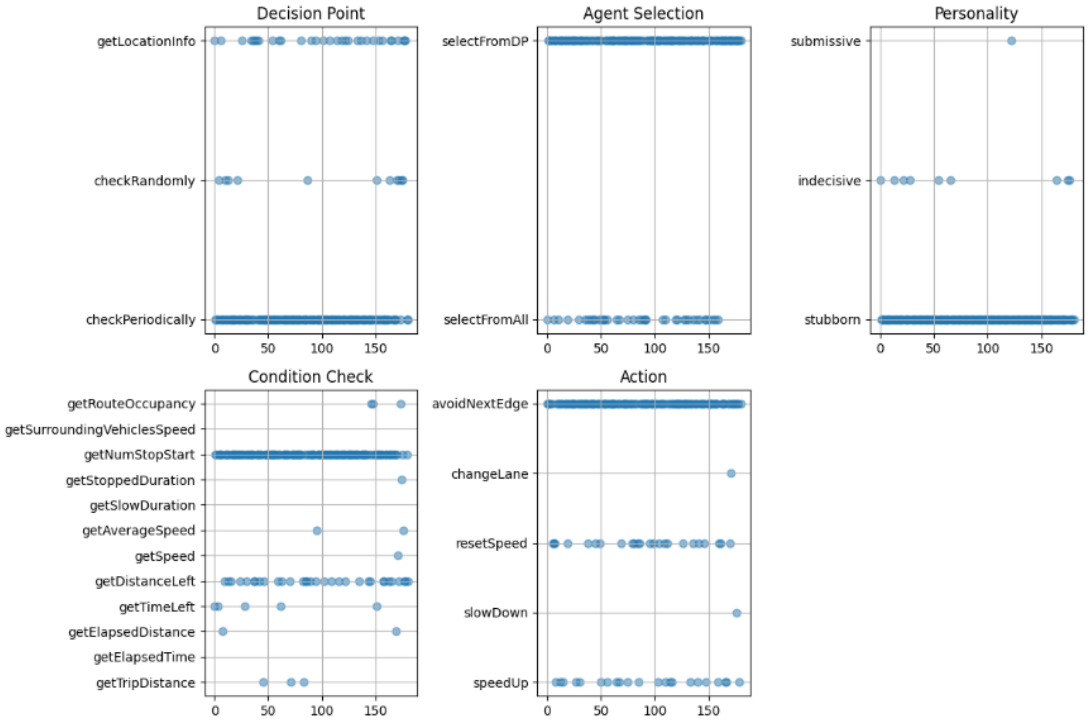

Figure B.7: Action Graph: Without Execution Block

B.2.3 Addition of Penalties

Action graph displaying an implementation stage when penalties were added to mitigate the previous issues. Although some changes can be seen, the problem persisted.

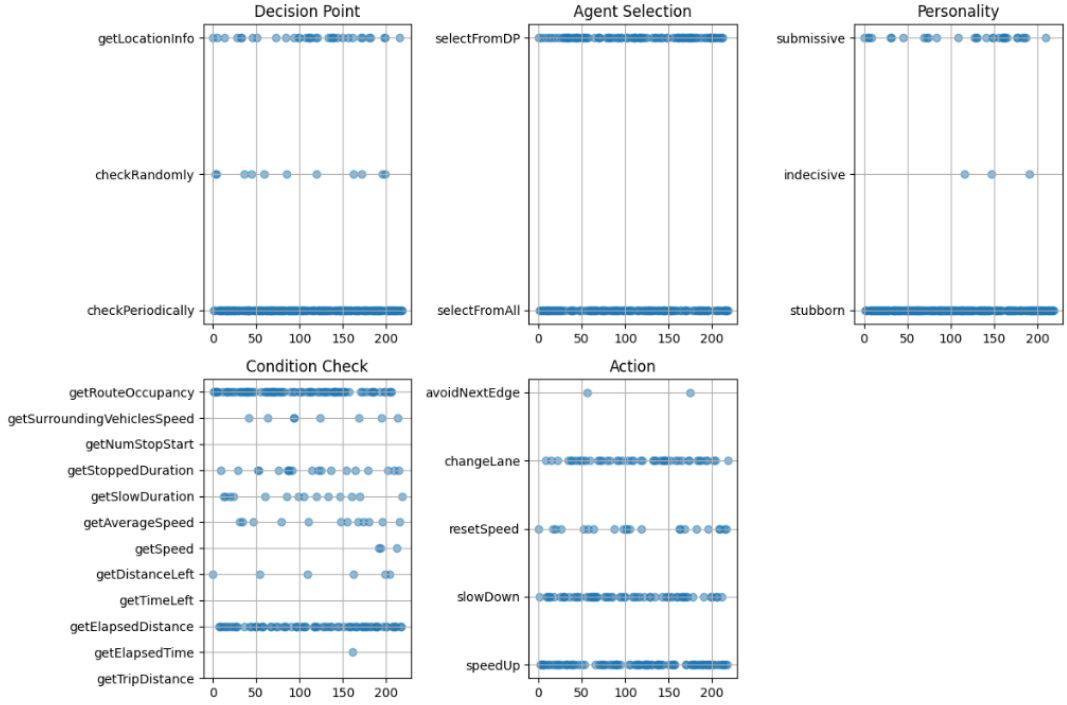

Figure B.8: Action Graph: Addition of Penalties

B.2.4 Creation of Value Options

Action graph displaying an implementation stage when value options for the block types were created.

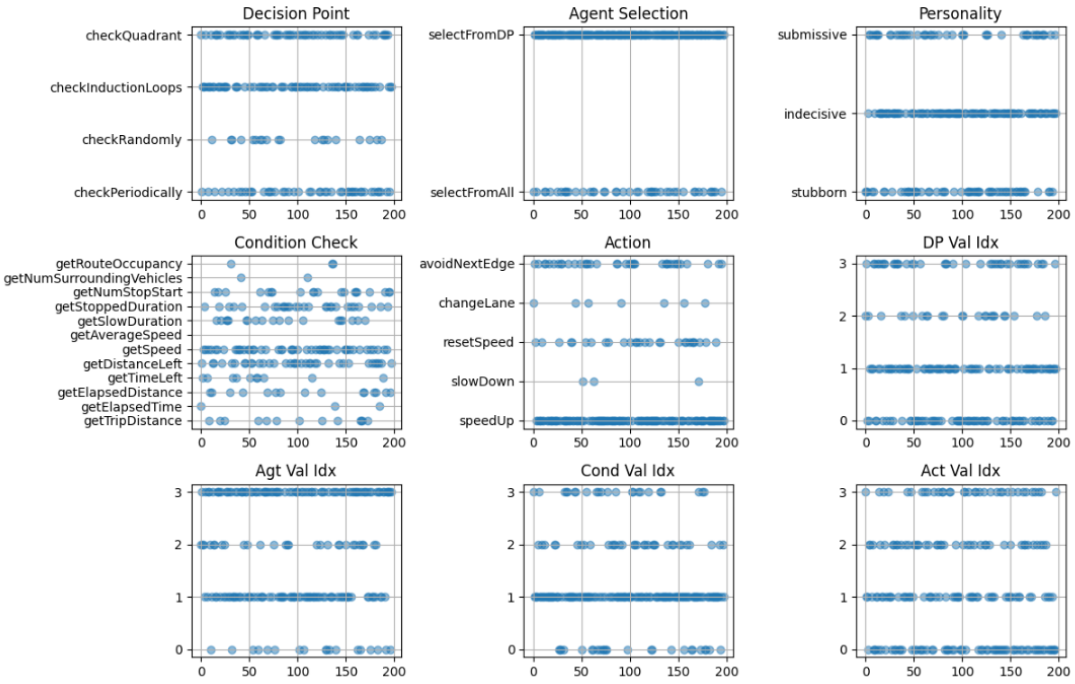

Figure B.9: Action Graph: Creation of Value Options

B.2.5 Colour-Coded Value Options

Action graph displaying the chosen value options in a colour-coded format for better visibility and understanding.

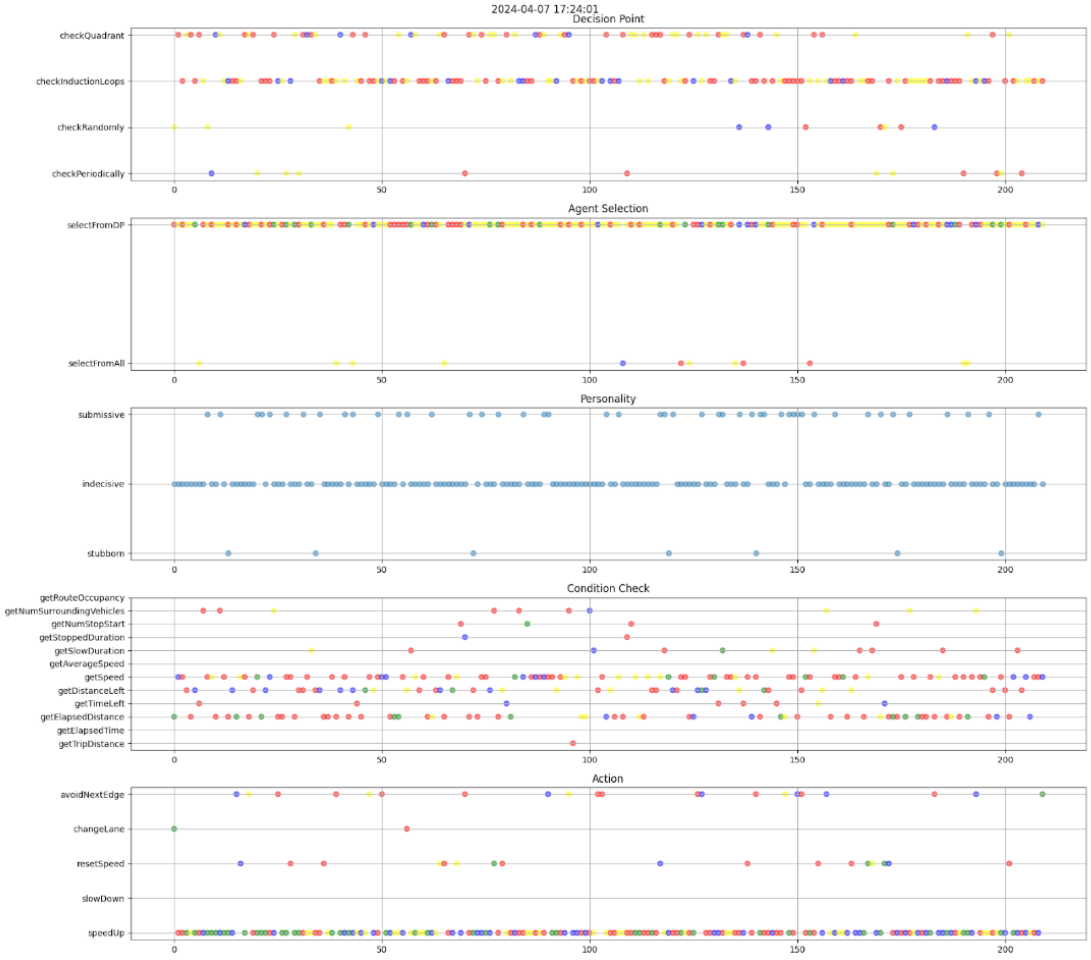

Figure B.10: Action Graph: Colour-Coded Value Options

B.2.6 Hyperparameter Tuning

Action graph showing a result where hyperparameter tuning was tested. Although this was the first result that returned a better scoring evaluation than the control group, it consisted of always selecting all the vehicles and increasing their speed, which is not considered valid or innovative for the purpose of this dissertation.

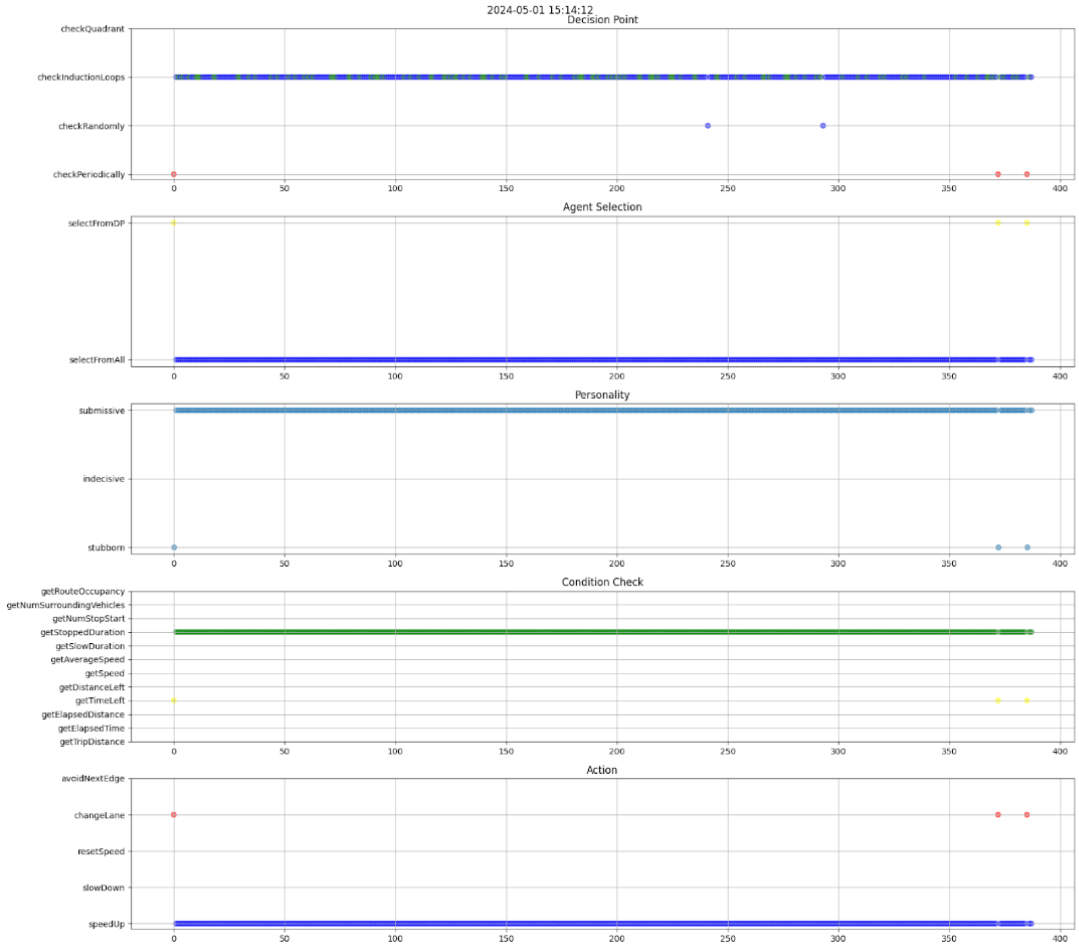

Figure B.11: Action Graph: Hyperparameter Tuning Result

B.2.7 Removal of "speedUp" and Addition of "enterHighway" and "addAvenue"

Action graph showing an implementation stage where the "speedUp" was removed. At this point, the new maps had been designed, so the "enterHighway" and "addAvenue" functions were created to make better use of the new network components.

Figure B.12: Action Graph: Removal of "speedUp" and Addition of "enterHighway" and "addAvenue"

After this result, several additional experiments were made. It was only upon the removal of the traffic lights that the model started returning valid positive results.

References

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A Brief Survey of Deep Reinforcement Learning. https://arxiv.org/abs/1708.05866v2, August 2017.

[2] Robert Axelrod. *The Complexity of Cooperation: Agent-Based Models of Competition and Collaboration*. Princeton University Press, August 1997.

[3] Steven C. Bankes. Tools and techniques for developing policies for complex and uncertain systems. *Proceedings of the National Academy of Sciences*, 99(suppl_3):7263–7266, May 2002.

[4] Thomas Bartz-Beielstein, Jürgen Branke, Jörn Mehnen, and Olaf Mersmann. Evolutionary Algorithms. *WIREs Data Mining and Knowledge Discovery*, 4(3):178–195, 2014.

[5] Michael Batty. *Cities and Complexity: Understanding Cities with Cellular Automata, Agent-Based Models, and Fractals*, volume 59. The MIT press, 2007.

[6] Ana L. C. Bazzan. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Autonomous Agents and Multi-Agent Systems*, 18(3):342–375, June 2009.

[7] Ana L. C. Bazzan and Ricardo Grunitzki. A multiagent reinforcement learning approach to en-route trip building. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 5288–5295, July 2016.

[8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, June 2009. Association for Computing Machinery.

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, New York, 2006.

[10] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of Data Science*. Cambridge University Press, 2020.

[11] Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. https://www.pnas.org/doi/epdf/10.1073/pnas.082080899.

[12] Daniel B. Botkin, James F. Janak, and James R. Wallis. Some Ecological Consequences of a Computer Model of Forest Growth. *The Journal of Ecology*, 60(3):849, November 1972.

[13] Dietrich Braess, Anna Nagurney, and Tina Wakolbinger. On a Paradox of Traffic Planning. *Transportation Science*, 39(4):446–450, November 2005.

[14] Helen Briassoulis. Analysis of Land Use Change: Theoretical and Modeling Approaches. 2020.

[15] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, June 2016.

[16] Keenan Burnett, Andreas Schimpe, Sepehr Samavi, Mona Gridseth, Chengzhi Winston Liu, Qiyang Li, Zachary Kroeze, and Angela P. Schoellig. Building a Winning Self-Driving Car in Six Months. https://arxiv.org/abs/1811.01273v1, November 2018.

[17] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A Comprehensive Survey of Multiagent Reinforcement Learning. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38:156–172, April 2008.

[18] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent Reinforcement Learning: An Overview. In Janusz Kacprzyk, Dipti Srinivasan, and Lakhmi C. Jain, editors, *Innovations in Multi-Agent Systems and Applications - 1*, volume 310, pages 183–221. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[19] Nicola Capodieci, Emma Hart, and Giacomo Cabri. Designing Self-Aware Adaptive Systems: From Autonomic Computing to Cognitive Immune Networks. In *Proceedings of the 2013 IEEE 7th International Conference on Self-Adaptation and Self-Organizing Systems Workshops*, SASOW '13, pages 59–64, USA, September 2013. IEEE Computer Society.

[20] Nicola Capodieci, Emma Hart, and Giacomo Cabri. Artificial Immunology for Collective Adaptive Systems Design and Implementation. *ACM Transactions on Autonomous and Adaptive Systems*, 11(2):6:1–6:25, June 2016.

[21] Rui P. Cardoso, Rosaldo J. F. Rossetti, Emma Hart, David Burth Kurka, and Jeremy Pitt. Engineering Sustainable and Adaptive Systems in Dynamic and Unpredictable Environments. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Distributed Systems*, volume 11246, pages 221–240. Springer International Publishing, Cham, 2018.

[22] Bo Chen and Harry H. Cheng. A Review of the Applications of Agent Technology in Traffic and Transportation Systems. *IEEE Transactions on Intelligent Transportation Systems*, 11(2):485–497, June 2010.

[23] Hussein Dia. An agent-based approach to modelling driver route choice behaviour under the influence of real-time information. *Transportation Research Part C: Emerging Technologies*, 10(5-6):331–349, October 2002.

[24] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking Deep Reinforcement Learning for Continuous Control, May 2016.

[25] T. B. Dutra, R. Marques, J.b. Cavalcante-Neto, C. A. Vidal, and J. Pettré. Gradient-based steering for vision-based crowd simulation algorithms. *Computer Graphics Forum*, 36(2):337–348, 2017.

[26] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*, volume 1. Cambridge university press Cambridge, 2010.

[27] Joshua M. Epstein. *Generative Social Science: Studies in Agent-Based Computational Modeling*. Princeton Studies in Complexity. Princeton University Press, Princeton, 2006.

[28] Paolo Fazzini, Isaac Wheeler, and Francesco Petracchini. Traffic Signal Control with Communicative Deep Reinforcement Learning Agents: A Case Study, December 2023.

[29] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning, May 2016.

[30] Richard M. Fujimoto. Research Challenges in Parallel and Distributed Simulation. *ACM Transactions on Modeling and Computer Simulation*, 26(4):22:1–22:29, May 2016.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[32] Volker Grimm, Eloy Revilla, Uta Berger, Florian Jeltsch, Wolf Mooij, Steven Railsback, Hans-Hermann Thulke, Jacob Weiner, Thorsten Wiegand, and Donald Deangelis. Pattern-Oriented Modeling of Agent-Based Complex Systems: Lessons from Ecology. *Science (New York, N.Y.)*, 310:987–91, December 2005.

[33] Emma Hart and Kevin Sim. A hyper-heuristic ensemble method for static job-shop scheduling. *Evolutionary Computation*, 24(4):609–635, December 2016.

[34] Dirk Helbing and Stefano Balietti. How to Do Agent-Based Simulations in the Future: From Modeling Social Mechanisms to Emergent Phenomena and Interactive Systems Design, October 2013.

[35] Zishuo Huang, Hang Yu, Zhenwei Peng, and Mei Zhao. Methods and tools for community energy planning: A review. *Renewable and Sustainable Energy Reviews*, 42:1335–1348, February 2015.

[36] Nishant Kheterpal, Kanaad Parvate, Cathy Wu, Aboudy Kreidieh, Eugene Vinitsky, and Alexandre Bayen. Flow: Deep Reinforcement Learning for Control in SUMO. In *SUMO 2018- Simulating Autonomous and Intermodal Transport Systems*, pages 134–115, 2018.

[37] Prabuchandran K.J, Hemanth Kumar A.N, and Shalabh Bhatnagar. Decentralized learning for traffic signal control. In *2015 7th International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–6, January 2015.

[38] Daniel Krajzewicz, Michael Bonert, and Peter Wagner. The open source traffic simulation package SUMO. *RoboCup 2006*, June 2006.

[39] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[40] Liza L. Lemos and Ana L.C. Bazzan. Combining adaptation at supply and demand levels in microscopic traffic simulation: A multiagent learning approach. *Transportation Research Procedia*, 37:465–472, 2019.

[41] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Ken Goldberg, and Ion Stoica. Ray RLlib: A Composable and Scalable Reinforcement Learning Library. *arXiv preprint arXiv:1712.09381*, 85, 2017.

[42] Pablo Alvarez Lopez, Evamarie Wiessner, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flotterod, Robert Hilbrich, Leonhard Lucken, Johannes Rummel, and Peter Wagner. Microscopic Traffic Simulation using SUMO. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2575–2582, Maui, HI, November 2018. IEEE.

[43] C. Macal and Michael North. Tutorial on agent-based modelling and simulation. *J. Simulation*, 4:151–162, September 2010.

[44] Stela Makri and Panayiotis Charalambous. Curriculum based Reinforcement Learning for traffic simulations. *Computers & Graphics*, 113:32–42, June 2023.

[45] Patrick Mannion, Jim Duggan, and Enda Howley. An Experimental Review of Reinforcement Learning Algorithms for Adaptive Traffic Signal Control. In *Autonomic Road Transport Support Systems*, pages 47–66. Springer, May 2016.

[46] John H. Miller and Scott E. Page. *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*. Princeton University Press, November 2009.

[47] Tom M. Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science. McGraw-Hill, New York, 1997.

[48] Anum Mushtaq, Irfan Ul Haq, Muhammad Azeem Sarwar, Asifullah Khan, Wajeeha Khalil, and Muhammad Abid Mughal. Multi-Agent Reinforcement Learning for Traffic Flow Management of Autonomous Vehicles. *Sensors*, 23(5):2373, February 2023.

[49] Kai Nagel and Fabrice Marchal. Computational methods for multi-agent simulations of travel behavior. *Proceedings of International Association for Travel Behavior Research (IATBR), Lucerne, Switzerland*, 2003.

[50] Truong Nguyen, Li Zhou, Virginia Spiegler, Petros Ieromonachou, and Yong Lin. Big data analytics in supply chain management: A state-of-the-art literature review. *Computers & Operations Research*, July 2017.

[51] Stephen W. Pacala, Charles D. Canham, and J. A. Silander Jr. Forest models defined by field measurements: I. The design of a northeastern forest simulator. *Canadian Journal of Forest Research*, 23(10):1980–1988, October 1993.

[52] Liviu Panait and Sean Luke. Cooperative Multi-Agent Learning: The State of the Art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, November 2005.

[53] Jan Peters. Policy gradient methods. *Scholarpedia*, 5(11):3698, November 2010.

[54] Jeremy Pitt, Dídac Busquets, and Sam Macbeth. Distributive Justice for Self-Organised Common-Pool Resource Management. *ACM Transactions on Autonomous and Adaptive Systems*, 9(3):14:1–14:39, October 2014.

[55] Jeremy Pitt, Josiah Ober, and Ada Diaconescu. Knowledge Management Processes and Design Principles for Self-Governing Socio-Technical Systems. In *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)*, pages 97–102, September 2017.

[56] Jeremy Pitt, Julia Schaumeier, and Alexander Artikis. Axiomatization of Socio-Economic Principles for Self-Organizing Institutions: Concepts, Experiments and Challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 7(4):39:1–39:39, December 2012.

[57] Dean A. Pomerleau. ALVINN: An Autonomous Land Vehicle in a Neural Network. In *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988.

[58] G. de O. Ramos and Ricardo Grunitzki. An Improved Learning Automata Approach for the Route Choice Problem. In Fernando Koch, Felipe Meneguzzi, and Kiran Lakkaraju, editors, *Agent Technology for Intelligent Mobile Services and Smart Societies*, pages 56–67, Berlin, Heidelberg, 2015. Springer.

[59] Bryan Raney, Nurhan Cetin, Andreas Völlmy, Milenko Vrtic, Kay Axhausen, and Kai Nagel. An Agent-Based Microsimulation Model of Swiss Travel: First Results. *Networks and Spatial Economics*, 3(1):23–41, January 2003.

[60] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67, November 2009.

[61] Rosaldo J F Rossetti, Ronghui Liu, Helena B B Cybis, and Sergio Bampi. A multi-agent demand model. In *Proceedings of the 13th Mini-Euro Conference and The 9th Meeting of the Euro Working Group Transportation*, pages 193–198, 2002.

[62] Rosaldo J.F Rossetti, Rafael H Bordini, Ana L.C Bazzan, Sergio Bampi, Ronghui Liu, and Dirck Van Vliet. Using BDI agents to improve driver modelling in a commuter scenario. *Transportation Research Part C: Emerging Technologies*, 10(5-6):373–398, October 2002.

[63] Stuart J. Russell, Peter Norvig, and Ernest Davis. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, Upper Saddle River, 3rd ed edition, 2010.

[64] Robert G. Sargent. Verification and validation of simulation models. In *Proceedings of the 2010 Winter Simulation Conference*, pages 166–183, December 2010.

[65] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1889–1897. PMLR, June 2015.

[66] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017.

[67] H. H. Shugart. *A Theory of Forest Dynamics: The Ecological Implications of Forest Succession Models*. Springer New York, November 1984.

[68] Kevin Sim, Emma Hart, and Ben Paechter. A Lifelong Learning Hyper-heuristic Method for Bin Packing. *Evolutionary Computation*, 23(1):37–67, March 2015.

[69] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018.

[70] Leigh Tesfatsion and Kenneth Judd. Handbook of Computational Economics, Vol. 2: Agent-Based Computational Economics. *Iowa State University, Department of Economics, Staff General Research Papers*, January 2006.

[71] Ivo J. P. M. Timóteo, Miguel R. Araújo, Rosaldo J. F. Rossetti, and Eugénio C. Oliveira. Using TraSMAPI for the assessment of multi-agent traffic management solutions. *Progress in Artificial Intelligence*, 1(2):157–164, July 2012.

[72] Evgeniya Ugnenko, Elena Uzhvieva, and Yelizaveta Voronova. Simulation of Traffic Flows on the Road Network of Urban Area. *Procedia Engineering*, 134:153–156, December 2016.

[73] H. Van Dyke Parunak, Robert Savit, and Rick L. Riolo. Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide. In Jaime Simão Sichman, Rosaria Conte, and Nigel Gilbert, editors, *Multi-Agent Systems and Agent-Based Simulation*, pages 10–25, Berlin, Heidelberg, 1998. Springer.

[74] Paul Waddell and Gudmundur Ulfarsson. Introduction to Urban Simulation: Design and Development of Operational Models. In *Handbook of Transport Geography and Spatial Systems*, volume 5, pages 203–236. Emerald Group Publishing Limited, 2004.

[75] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[76] Axel Wegener, Michał Piórkowski, Maxim Raya, Horst Hellbrück, Stefan Fischer, and Jean-Pierre Hubaux. TraCI: An interface for coupling road traffic and network simulators. In *Proceedings of the 11th Communications and Networking Simulation Symposium*, pages 155–163, Ottawa Canada, April 2008. ACM.

[77] Uri Wilensky and William Rand. *An Introduction to Agent-Based Modeling: Modeling Natural, Social, and Engineered Complex Systems with NetLogo*. The MIT Press, 2015.

[78] Hai Yang and Michael G. H. Bell. Models and algorithms for road network design: A review and some new developments. *Transport Reviews*, 18(3):257–278, July 1998.

[79] Lun Zhang, Wenchen Yang, Jiamei Wang, and Qian Rao. Large-Scale Agent-Based Transport Simulation in Shanghai, China. *Transportation Research Record*, 2399(1):34–43, January 2013.

[80] M. Zhang. *Large-Scale Agent-Based Social Simulation: A Study on Epidemic Prediction and Control*. PhD thesis, Delft University of Technology, 2016.

[81] Wei Zhang, Andrea Valencia, and Ni-Bin Chang. Synergistic Integration Between Machine Learning and Agent-Based Modeling: A Multidisciplinary Review. *IEEE Transactions on Neural Networks and Learning Systems*, 34(5):2170–2190, May 2023.