Congestion Control with Collaborative Diffusion

Computer Science (Software Engineering)
Supervisor: Dr Gary Ushaw
May 2020

Deyan Samardzhiev

Abstract

In this project, collaborative pathfinding is explored as a means of improving the efficiency of smart parking systems as a way to reduce the overall congestion in urban environments. A significant amount of traffic congestion is caused by drivers looking for parking spaces. For this reason urban areas employ parking guidance and information systems to reduce congestion. The most common solutions to navigation in these systems rely on derivatives of A* pathfinding, which does not take account of multiple agents navigating towards the same goals at the same time. By expanding on previous research, this project aims to validate and improve collaborative diffusion as an alternative to its non-collaborative counterpart. A simulation of a smart parking system is created for the Newcastle upon Tyne area, where multiple agents attempt to collaboratively reach their goals and reduce traffic.

Declaration

“I declare that this dissertation represents my own work except where otherwise stated.”

Acknowledgements

Firstly, I would like to thank Dr Gary Ushaw for allowing this project to be undertaken. He has provided continued advice and support throughout my studies in Newcastle University. Thanks also to Dr Graham Morgan and Dr Rich Davison for their encouragement and advice!

Table of Contents

Chapter 1: Introduction

1.1 Purpose

In recent years, drivers searching for parking spaces have become one of the main contributors to road congestion. They account for 30-50% of the overall urban congestion [1] [2]. The commonly used older solutions for optimising navigation and reducing traffic, such as bigger parking spaces, smarter roads and park and ride systems, have proven largely unsuccessful [2] [3]. As an alternative, local authorities implement parking guidance and information (PGI) systems, providing drivers with information on the availability and location of parking spaces.

PGI systems allow for smart parking structures based on resource sharing [4] [5] [6]. These schemas can rely on the information available about free parking spaces and pre-made reservations. When a goal has been chosen, the driver is navigated towards the advised parking space, usually using GPS. When multiple agents are involved, solutions become exponentially more problematic because they could all have the same goal or colliding paths, which can cause congestion. Congestion control is now an active area of research and various systems have been proposed that take account of current traffic and road accidents using different available resources, such as 4G, 5G or smart sensors [7] [10].

The utility of PGI systems is evident through the reduction of noise (decrease in the number of vehicles) and exhaust emissions (shorter travel times) [2]. When employed correctly, they become an asset to a metropolitan area. It is therefore important to further develop these systems.

In this paper, route planning for PGI is explored using collaborative diffusion (CD), with the goal of reducing congestion. CD is a collaborative pathfinding algorithm that uses its environment to communicate shortest paths [8]. These paths are created so as not to interfere with other agents' paths. An adaptation of this algorithm is also proposed, which allows for the control of "conformity" of route choices. A simulation of Newcastle City Centre is presented, where agents simultaneously navigate towards their goals, while taking account of the traffic. In the simulation, these agents represent the drivers and the goals – the parking lots. A grid system, created by nodes and edges, represents the Newcastle upon Tyne urban road system.

1.2 Project Aims and Objectives

This dissertation aims to evaluate collaborative diffusion as a means of alleviating congestion. The aim is divided into the following objectives:

1. Research algorithms and smart parking: this objective includes researching solutions for collaborative pathfinding and the area of smart parking systems. Collaborative diffusion is also thoroughly researched.

2. Simulate congestion: this objective aims to create an environment capable of simulating congestion by implementing the research. The original aim of the project

shifted from creating such an environment to adapting an existing framework. The goal is divided into two sub-goals:

   a. The testing environment: create an automated way of visualising the pathfinding algorithms and their effects;

   b. The simulation: the main aim of the project is to show an implementation of collaborative diffusion within a PGI system. To achieve this, a mid-fidelity model has to be built to simulate this congestion. In this project the model is based on the Newcastle upon Tyne metropolitan road system.

3. Algorithm augmentation: the aim is to propose an augmentation to the collaborative diffusion algorithm. This goal was added during the development of the project, when the collaborative diffusion faced a bottleneck issue (described in Chapter 3: Design and Implementation). The solution was inspired by the behaviour of diffusion and mass flow in physical systems [9].

4. Evaluation and statistics tools: the last step in the dissertation is assessing whether the simulation captures congestion and CD reduces it. To be able to assess this, standard metrics of congestion have to be researched and the tools for extracting this data from the simulation need to be implemented.

Chapter 2: Background Research

2.1 Research Strategy

The research undertaken was divided into three main areas:

1. Smart parking: this includes research on the topic of PGI systems, navigation algorithms and limitations. This topic is typically related to smart cities, and the overarching theme is referred to by researchers as IoT (internet of things). Research papers on this topic were found through Google Scholar, the university library and by following cited works in publications.

2. Algorithm research: collaborative and non-collaborative pathfinding solutions were researched and evaluated. Diffusion is explored both as a concept and as a specific algorithm. The papers and publications cited were in the context of either pathfinding solutions for games or navigation. Physics and discrete mathematics textbooks were also researched.

3. Pathfinding framework: a past project containing an implementation of collaborative diffusion and a road network of the Newcastle upon Tyne metropolitan area was used in the creation of this project. The framework is described in this section.

2.2 Background Research

2.2.1 Smart Parking

In most cities public parking is viewed as a source of revenue rather than a cost [12], this is clearly shown in Figure 2.1. As cities become increasingly populated and congested, controlling and optimising public parking becomes ever more a key issue with an economic impact, however. Interestingly, 100% of surveyed Parisian drivers have abandoned a trip or parked illegally because of the annoyance of finding a parking space [13]. Having information about the availability and reservation of parking spaces in real-time would alleviate this problem, hence smart parking systems have evolved as a solution.

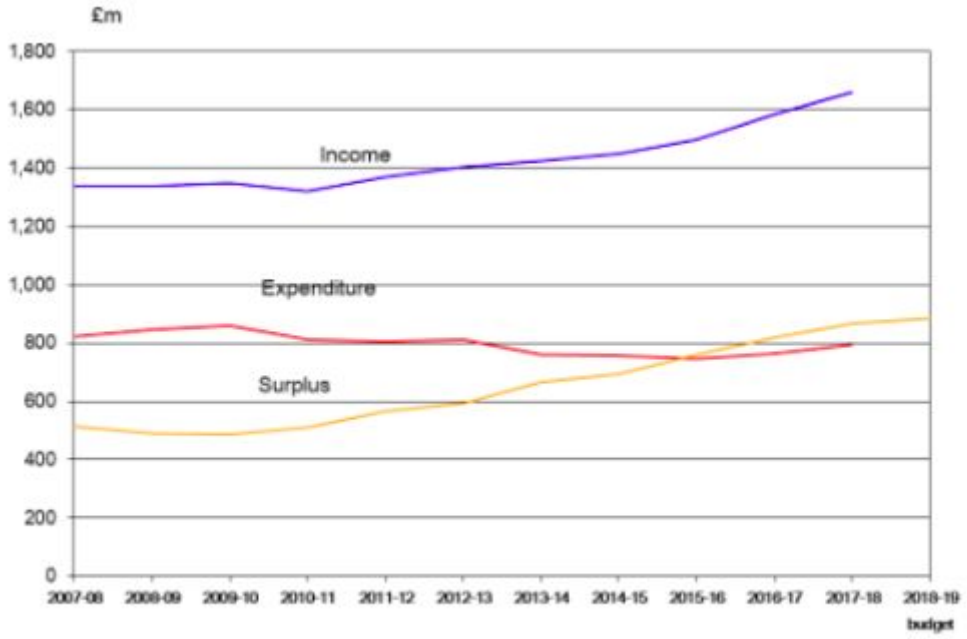


Figure 2.1 Parking income, expenditure and surpluses for the UK. [12]

“Smart parking is a way to help drivers find more efficiently satisfying parking spaces through information and communications technology” [13]. These systems can create an overall net advantage as they lead to improved parking space utilisation and market-sensitive pricing. Using this definition, PGI can be viewed as part of the infrastructure for smart parking systems. As noted above, PGI systems have become a vital part of the metropolitan areas that employ them as they reduce overall traffic and environmental effects [2] [4]. They present drivers with dynamic information on parking and direct them to vacant parking spots, usually using GPS.

There are inherent issues with a straightforward approach to navigating directly to a parking space, however. By guiding many drivers to an area with few vacant spots, congestion can be caused. The problem becomes even worse when the planned routes for those vehicles are the same or cross over with each other. In early iterations of these PGI systems, the pathfinding algorithms used did not take account of these issues, causing undue congestion.

More recently, the PGI technology has allowed for the introduction of reservation systems. A driver is allocated a specific place, which is shown as unavailable until that specific driver arrives [5]. The topic of resource allocation is widely researched and algorithms such as PIS have optimised solutions to a significant extent [6] [14]. However, only limited research has been done on how to route vehicles to parking spaces within the PGI environment, with the goal of reducing congestion.

2.2.2 Collaborative Pathfinding

Pathfinding algorithms are used to navigate autonomous agents in their environment. Typically, the environment is represented as nodes (i.e. traversable points in space) and the connections between these nodes are represented as edges. Nodes connected to other nodes via edges are called neighbours. At each node the agent can only travel to the direct

neighbour of the current node it is residing on. In this paper this is referred to as a grid or graph and the autonomous agents as avatars or agents.

The problem of navigating single agents is widely explored in the fields of both mathematics and computer science. There is a variety of solutions optimised for different environments and requirements. These include algorithms such as Djikstra's algorithm, A*, Breath-First, Depth-First search, IDA* and ACO [15] [16]. A visual comparison is presented in Figure 2.2.

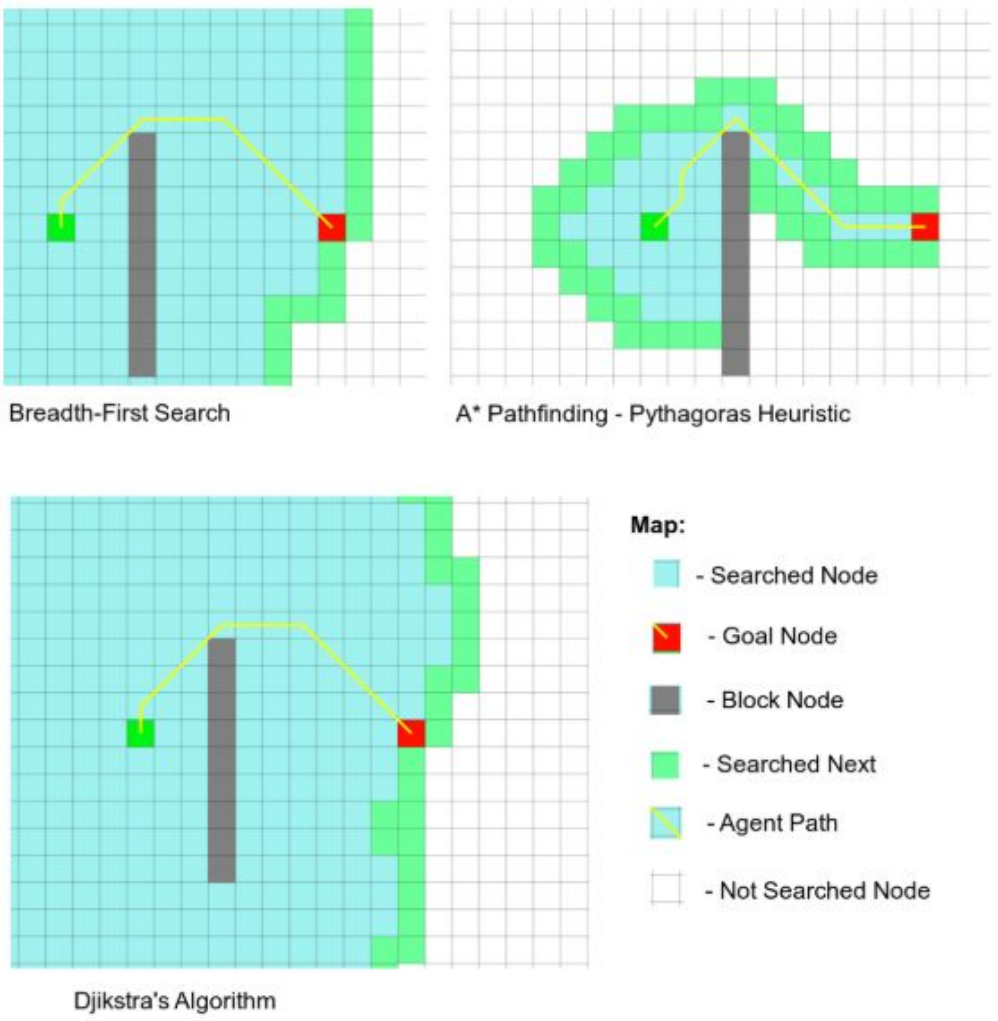


Figure 2.2 Visual comparison of A*, Bread-First Search and Djikstra pathfinding
Generate by https://qiao.github.io/PathFinding.js/visual/

Navigation of multiple agents with potentially conflicting goals in a shared environment is an area of ongoing research. Multi-agent path planning has been shown to be a PSPACE difficult problem. Possible states expand exponentially with the increase in the number of agents [17] [18]. The path planning itself is not the issue as it is known to be parallelisable [21]; the actual problem relates to relaying path information to and from each of the agents.

One of the few researchers looking into this area is David Silver, who has proposed three algorithms based on A*: CA*, which searches for non-colliding routes between agents;

HCA*, which boosts the performance of CA*; and WHCA*, which limits the search time/space of HCA* [19]. This work is further developed by Rhodes et al. [20], where a novel conflict-based approach to the problem is established.

The general idea of the algorithms is to sacrifice optimal solutions to create non-colliding paths between the agents. This collaborative planning prevents congestion. The solutions could potentially be implemented in a PGI system and utilise GPS as a shared environment to communicate paths and positions. This would, in theory, substantially reduce congestion.

2.2.3 Collaborative Diffusion Pathfinding

Another recent development in the field of multi-agent pathfinding was made by Reppening [8], who introduced the collaborative diffusion pathfinding algorithm. The concept of diffusion itself is a well-known physical process. It describes the spread of matter from areas of high concentration to areas of low concentration over time.

The idea of pathfinding using diffusion has been previously researched using cascading models [24] (i.e. based on probability calculations). Diffusion is also commonly used to analyse graphs [25] and propagation of information and social behaviours [26]. The contribution of the collaborative diffusion algorithm comes from the interaction between agents. The idea was introduced alongside the concept of anti-objects, as an alternative to traditional pathing algorithms. Use of anti-objects, as the name suggests, entails conceptualising object roles in a different manner. “Antiobjects allow us to literally think outside the proverbial box or, in this case outside the object” [8].

For example, in the game Pac-Man the ghost and floor tiles are examples of anti-objects. Software engineers typically see the floor tiles as passive while the ghost agents are active. For anti-objects these roles would be reversed. This results in a shift in the pathfinding computational requirements from the seemingly active agents to the environment. Agents appear to be active and intelligent, but in reality they become reactive to the topology of the environment and location of their goal (seen in Fig. 2.3), which is the basis of collaborative diffusion pathfinding.

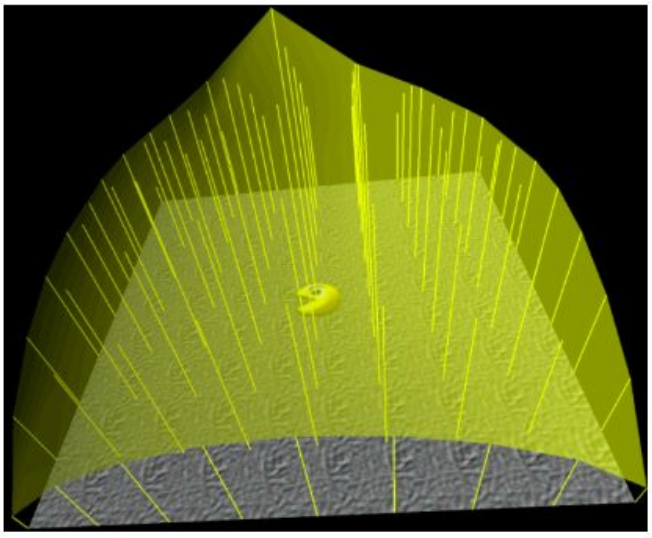


Figure 2.3. Worksheet with 9 x 9 floor tiles and one Pacman agent in the center.
The Pacman "scent" is diffused by the tiles through the field.
These values are subsequently used to navigate to the Pacman agent [8]

In collaborative diffusion pathfinding, the "scent" of a goal is spread through a grid system. The goal is given an arbitrary positive or negative constant value, which indicates its desirability in relation to other goals. Negative values will be avoided; positive ones will be pursued. This value is then diffused through the field using Equation 1.

Equation 1

$$u_{0,t+1} = u_{0,t} + D\sum_{i=1}^{n}\left(u_{i,t} - u_{0,}\right)$$

Where:

n = number of neighbouring agents used as input for the diffusion equation

$u_{0,t}$ = diffusion value of centre agent

$u_{i,t}$ = diffusion value of neighbour agent (i > 0)

D = diffusion coefficient [0..0.5]

The diffusion coefficient controls the speed of diffusion; larger values will result in a quicker spread of the "scent". An example diffusion pattern can be seen in Figures 2.3 and 2.4. The cost of single-step value calculations increases linearly O(n) with the number of direct neighbours for each node.

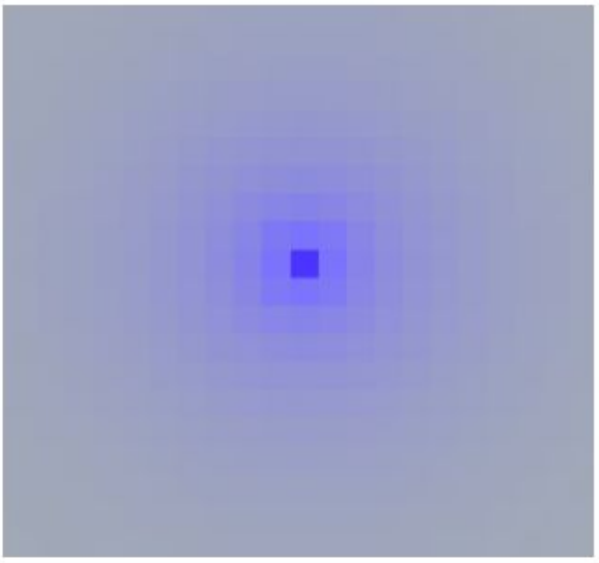


Figure 2.4. Diffusion pattern generated from the simulation.
The node in the center is the goal node. Its value is diffused across the field.
The more saturated a particular spot is the greater its diffusion value.

Note that Equation 1 was derived by Reppening from a paper researching physically-based visual simulations on graphics hardware. The implementation used in this paper was [22]:

Equation 2

$$T'_{i,j} = \frac{1}{4}\sum_{k=1}^{4}\left[\left(1 - c_d\right)T_{i,j} + c_d T_{n_k(i,j)}\right]$$

Where:
$T_{i,j}$ = a two-dimensional scalar field
n_k(x,y) = the kth nearest neighbour of (x, y)
c_d = the diffusion coefficient

In Equations 1 and 2 the authors assume the algorithm to be used in a matrix rather than a graph [8]. In particular, this involves assuming that a node has four immediate neighbours, defined by a von Neumann neighbourhood [23]. In the form above, it is seen that the diffusion operator is the average of four weighted sums of the centre texel, Ti, j and its four nearest neighbour texels [22]. Equations 1 and 2 do not take account of different edge weightings or direction, and they would potentially be wrong for a node with more than four neighbours. This is further expanded on in the ‘Design and Implementation’ section.

Once the goal value has diffused across the grid, with simple hill-climbing, at any node, agents can backtrack the shortest path to that goal. The same equation (Eq.1) works for multiple goals. They may have different values representing desirability, and diffusion patterns are created using both the distance of the goals and their desirability.

Agents passing through a node directly influence the diffusion value of that node. This produces an emergent behaviour where agents interact with each other in different ways.

Interaction can be either collaboration (agents increase the diffusion value of the node they pass) or competition (agents reduce the diffusion value), or neither (no interaction). An example of emerging competitive behaviour is agents avoiding each other. This is implemented by adapting the previous formula to:

Equation 3

$$u_{0,t+1} = \lambda \left(u_{0,t} + D \sum_{i=1}^{n} \left(u_{i,t} - u_{0,} \right) \right)$$

Where:
n = number of neighbouring agents used as input for the diffusion equation
$u_{0,t}$ = diffusion value of centre agent
$u_{i,t}$ = diffusion value of neighbour agent (i > 0)
D = diffusion coefficient [0..0.5]
λ = agent interaction variable

The additional λ is used to manipulate the diffusion value and therefore the interaction between agents. When the formula uses $\lambda < 1,$ the emergent behaviour is competition between agents and, conversely, when $\lambda > 1$ is used, collaboration emerges. A list of behaviours for the different values of λ can be seen in Table 2.1. Agents passing over the nodes can set the values of λ and directly affect the overall behaviour.

Table 2.1 Agent interaction [8]

λ	**Agent Interaction**
>> 1	Extreme Collaboration
> 1	Collaboration
1	Autonomy
<1	Competition
<<1	Extreme Competition

The simplicity of the formula makes collaborative diffusion pathfinding “embarrassingly parallel” [21]. The problem can be split into a number of parallel tasks that are compatible with GPU calculations. In non-uniform environments (i.e. edges do not all have the same values) diffusion typically finds shorter paths than A* or Dijkstra. In environments containing obstacles, diffusion finds results comparable to A* in terms of cost and in a time-scale appropriate to real-time path finding (Fig. 2.5 A* versus diffusion). Note that through the

implementation of the diffusion formula, this paper uses a weighted tile map and might not be representative for all data structures.

Another benefit of the algorithm is that it does not have to recompute paths every time there is an environmental change. This task is handled in a distributed manner by the environment itself. This is not the case for A*, which has to re-compute its path.

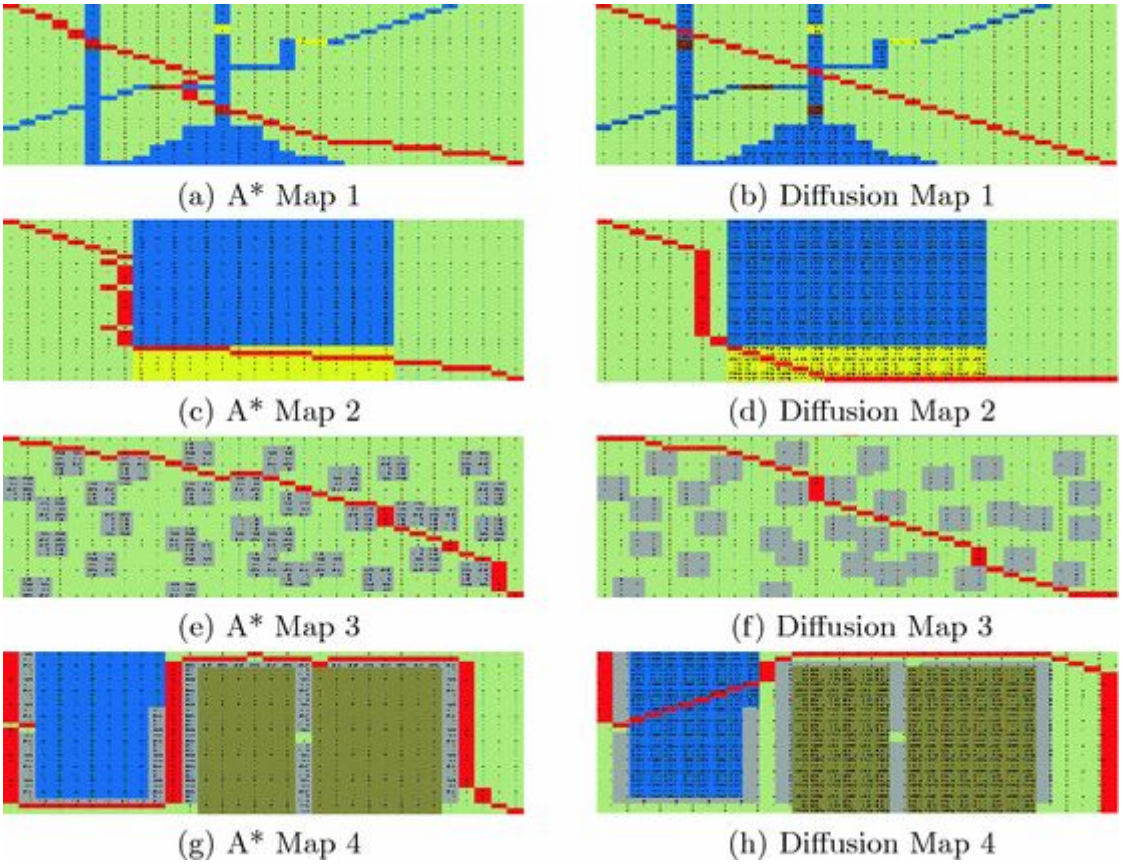


Fig. 2.5 A* versus diffusion [21]. The different terrains have different edge costs. Blue terrain water has a higher cost than yellow terrain. Yellow has a higher cost than light green and grey and dark green tiles represent impassable obstacles.

2.2.4 Diffusion

To achieve better appreciation of what the algorithm is doing, diffusion is briefly explored as a physical process. Diffusion as a process refers to movement against a concentration gradient. It is characterised by the movement of mass from regions of high to low concentration [27]. This describes, for example, the net flow of particles from a chemical dropped in still water or the distribution of heat in a solid body through time. The equation that describes this movement is as follows:

Equation 4 [28]

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u + s$$

Where:

u = a scalar field representing the diffusion concentration gradient (e.g. the distribution of heat)
a = diffusion constant
$\nabla^2 u$ = the Laplacian of a scalar field
s = scalar field representing a source or sink

Note: The Laplacian is a scalar operator that takes in and returns a scalar. It is defined by the divergence of the gradient of a scalar function. It is equivalent to the second derivative of a scalar function.

This specifically describes the change in distribution with a change in time. The u term is a scalar field representing the distribution itself (in our case this would be the tile or node grid itself) and α is the diffusion constant describing the rate of diffusion in the body, i.e. the diffusion coefficient [27] [28]. In material science each material has a specific heat diffusion coefficient. Silver, for instance, has a much larger heat diffusion coefficient than a thermal insulator such as wood. Consequently, silver will spread heat much faster than wood. The term s refers to the scalar field of heat sources (a positive value of s indicates a source and a negative value suggests a sink) [28].

This gradient is best understood when visualised. The image in Figure 2.6 presents such a concentration gradient. A higher concentration value is represented by the darker shade of purple. The object at the centre of the image is known as a source, i.e. the region that increases the concentration. An object that decreases the concentration would be known as a sink [29].

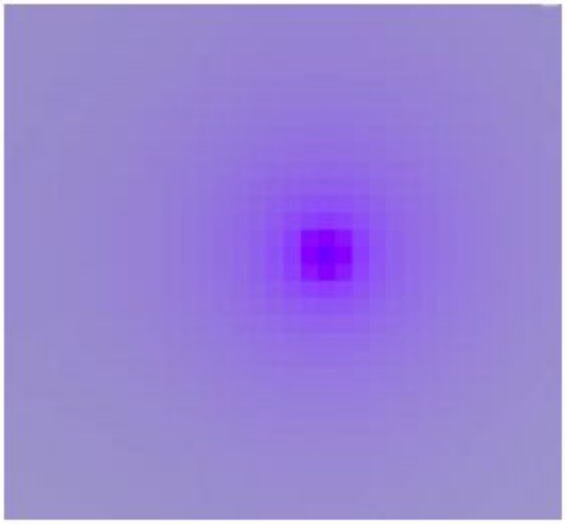


Figure 2.6. Diffusion Gradient

Note that diffusion represents the movement of individual particles rather than the whole flow of particles. Convection (or advection) describes the flow or bulk motion of the particles [28]. The two terms come together when describing convection-diffusion, which describes the physical phenomenon of particles moving by both their own individual movement and the bulk flow. Simulating these phenomena in fluids is an active field of research and different solutions have been proposed, mainly based on the Navier-Stokes equations [32].

To draw an analogy with collaborative diffusion, the concentration gradient (diffusion) is essentially iteratively calculated at each time-step in Equation 1 using a Gauss-Seidel method [30]. Simply put, the next $(u_{i,t+1})$ is computed using the value at the previous time-step $(u_{i,t})$ [31] [22].

The algorithm described in Equations 1 and 2 depend on the structure it is implemented on. In particular, it requires the 'field' to be represented by a f(x,y) function or a scalar field and for each 'node' to spread/take its diffusion value to/from four neighbours. This is due to the fact that Equation 1 was originally derived from its continuous version:

Equation 5 [22]

$$T'_{i,j} = T_{i,j} + \frac{c_d}{4} \nabla^2 T_{i,j}$$

Where:
$T_{i,j}$ = a two-dimensional scalar field
c_d = the diffusion coefficient
$\nabla^2 T_{i,j}$ = the Laplacian of a scalar field

Note: A direct parallel can be drawn between the heat equation (Equation 4) and the one presented in the original paper [22] (Equation 5).

Equation 5 was rewritten using the discretised form of the Laplacian operator:

Equation 6 [22]

$$\nabla^2 T_{i,j} = T_{i-1,j} + T_{i+1,j} + T_{i,j-1} + T_{i,j+1} + 4T_{i,j}$$

The values of each of the adjacent positions in the scalar field ($T_{i,j}$) were then replaced with the nearest four neighbours of a texel, leaving us with Equation 2. The main point raised here is that the computation is strictly based on four texels at a time, without the inclusion of direction or weightings. This is further elaborated in Chapter 3: Design and Implementation.

2.2.5 Pathfinding Framework

This project builds on a previously developed framework that shared the same goal of assessing collaborative diffusion as a means of congestion control. This framework was developed using the Unity 3D Engine. It originally consisted of a testing environment, a simulation representing the Newcastle upon Tyne road system and the implementation of CD, Djikstra and A*. The decision to build upon an older project, rather than creating a new one, was made because of the similarities in structure, aim, programming environment choice and algorithms researched. This project attempts to further develop this framework.

The past framework is described and the bedwork for the current projects' specifications are laid out. The original project is referred to as the pathfinding framework. It is best described in three parts: the structure, the environment and the algorithms employed.

The structure

The framework is built using a solid node system that can be presented as a weighted directed graph. The three main classes used in this graph are: node, edge and GridManager.

The node class represents a vertex in a graph. It stores information about its outgoing edges, cost, its current diffusion values (separate from the cost) and the status of the node. The status is a value indicating whether the node is a starting point, an end point or a blocked node. It stores diffusion values that are constantly updated using Equation 2.

The edge class represents an edge pointing from a parent node to a neighbour node. It stores information about which node it is pointing to, which node it is coming from and the cost. It allows the node to access information about its own neighbourhood.

The GridManager class manages the grid (or graph) and is responsible for all tasks that are outside the scope of single nodes or edges. It stores an array of nodes and is responsible for scheduling their updates.

The AI agents navigating the graph are implemented using a DiffusionAvatar class. These objects are spawned, updated and removed by the GridManager. If the GridManager's current pathfinding algorithm is set to diffusion agents it will individually compute paths. Otherwise, the objects will be moved along paths directly by the GridManager. After a DiffusionAvatar reaches its goal it flags itself for removal by the GridManager.

The main computation in this whole structure might seem to be in the GridManager. This is indeed the case when it comes to A* and Djikstra pathfinding, but for diffusion the calculations are shifted toward the individual nodes of the grid. The in-depth relationship between these classes can be seen in Figure 2.7, showing the class UML diagram.

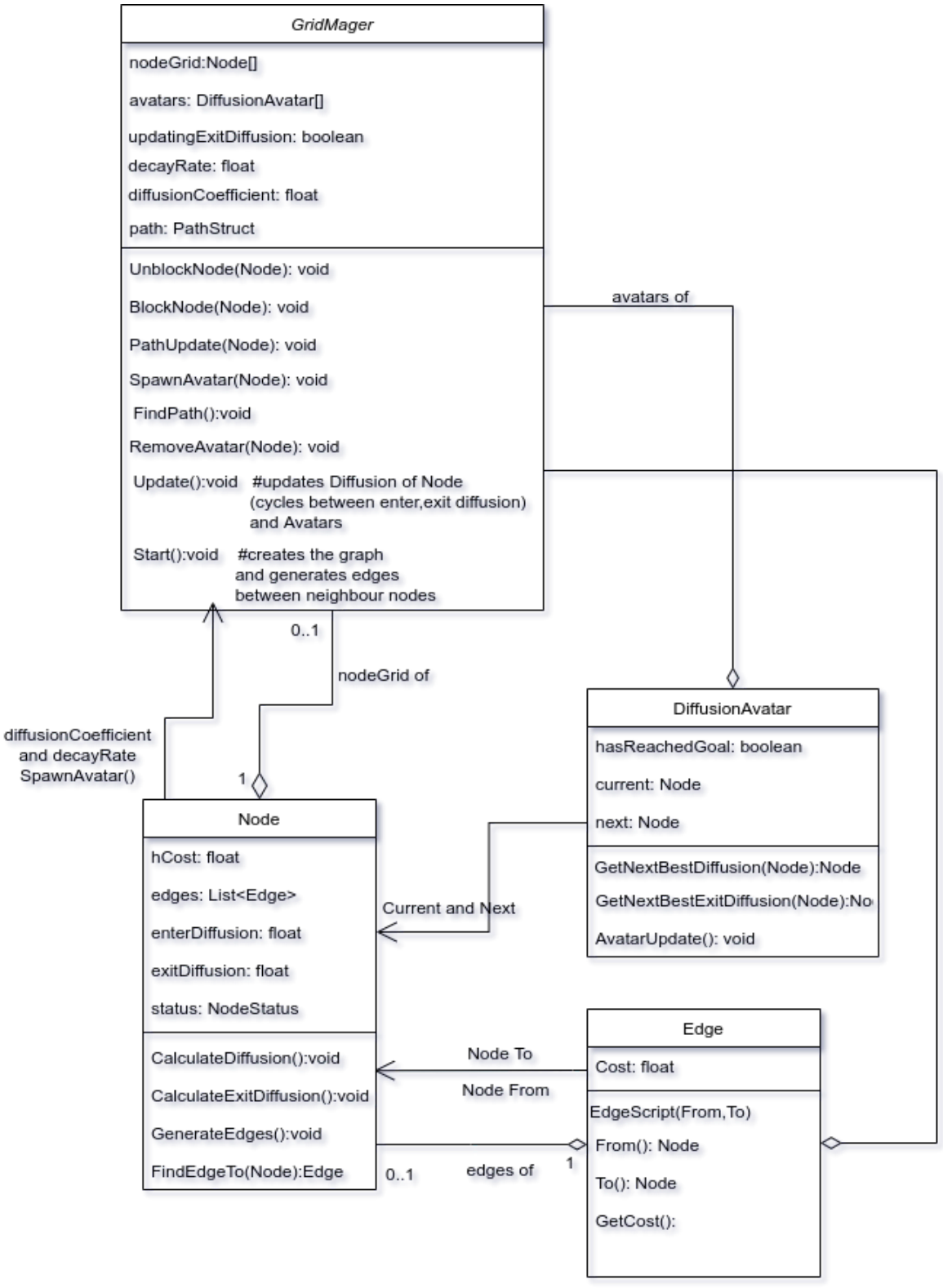


Fig. 2.7 Class UML diagram

The environments

The project consisted of two unity scenes, one of them a 16x16 grid, each one connected to its immediate neighbour (maximum of eight, including diagonally). This is referred to as the testing environment. The other one was a simulation representing the Newcastle upon Tyne road network as a grid. This is referred to as the Newcastle simulation.

Testing environment

In the testing environment a user can block nodes to create different maps on the grid (as seen in Figs 2.8 and 2.9). Autonomous agents can be spawned by clicking on nodes and the user can cycle between the pathfinding algorithms.

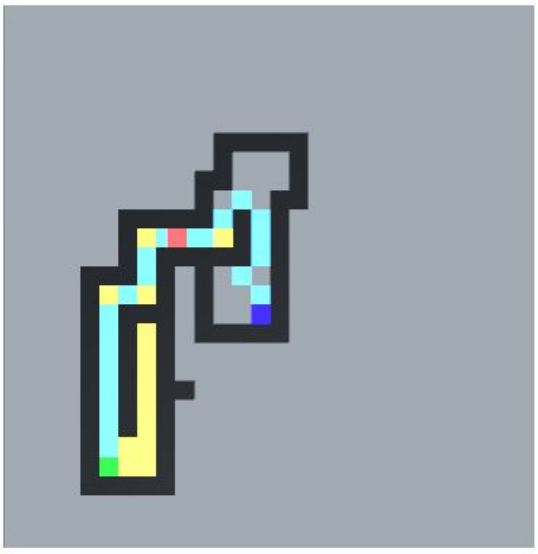


Figure 2.8. Testing Environment Grid for Non-Collaborative solutions. The light blue nodes represent the path of the agent, the yellow ones are searched nodes (by A* or Djikstra). The red node is a agent. The black nodes are blocked(i.e. the agent cannot pass through them) and the purple node is the goal

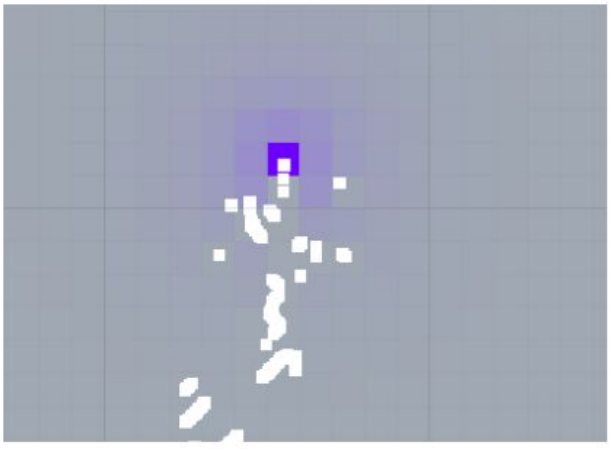


Figure 2.9 Testing Environment Grid for Collaborative Diffusion. The smaller white boxes are agents, and the purple node is the goal. The saturation of nodes represents their diffusion values.

The variables of these algorithms are easily editable through the Unity UI (Fig. 2.10). The user is also able to switch between the different algorithms using the specified hotkeys. This scene is the main place for testing ideas and visualising the effects of diffusion.

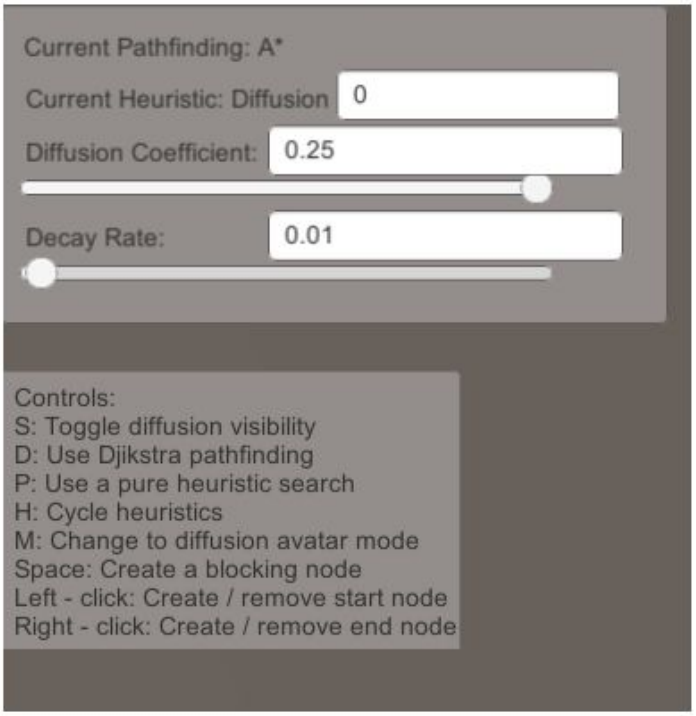


Figure 2.10 Testing Environment Controls.
The sliders and the input boxes directly change the global variables for the diffusion computation.

Newcastle simulation

In the Newcastle simulation the DiffusionAvatar objects can only use CD pathfinding. The A* and Djikstra solutions have not been implemented in this scene, presumably because of the high computation costs.

Agents can be seen coming in, parking for some time and then leaving. This is achieved by having two diffusion values for the two separate goals. The avatars follow the “scent” of either the “exit” diffusion or the “enter” diffusion (i.e. a parking node). When they reach a parking node, they are marked for removal. At this point the parking lot schedules the task of spawning another avatar that follows the exit diffusion path after a random amount of time. This simulates the behaviour of people coming into the city, parking for some hours and then leaving, as seen in Figure 2.11. Through this agent behaviour, statistics of the revenue generated from parking are displayed in UI window 2.12.

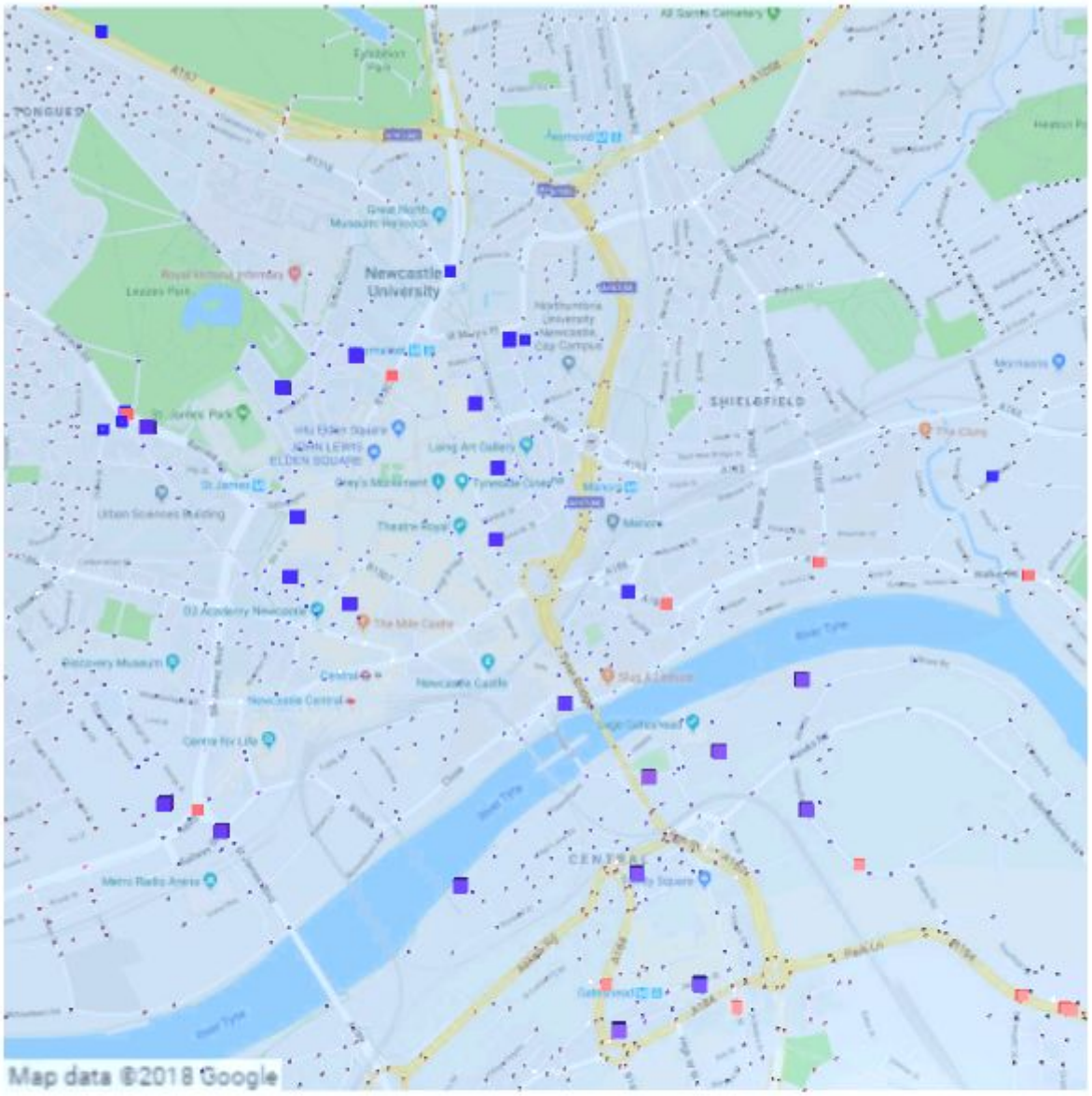


Figure 2.11 The Newcastle Simulation.
The smallest dots seen on the map are the nodes. They are position and connected so as to represent the road network. The large blue squares are parking lots. The smaller blue squares are the agents entering and parking, and the red squares are the agents leaving the simulation.

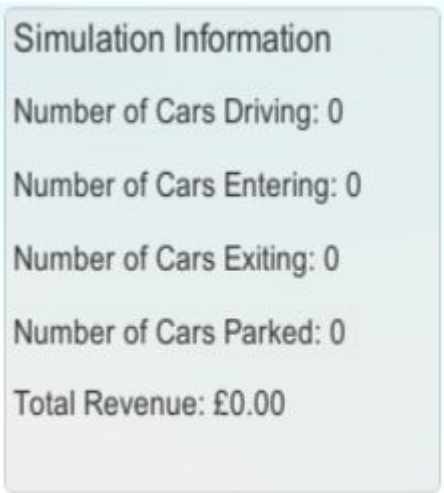


Figure 2.12 Parking Statistics

The user can also change the speed of the agents, the evasion strength (a multiplier for the agent interaction factor – λ) and the spawn rate of agents.

The simulation spawning works differently from the testing environment. Designated nodes are marked as “spawner” nodes. In each update these nodes randomly decide to task the GridManager to spawn

a DiffusionAvatar on them. Spawning is uniformly random and is scaled by the spawn rate variable.

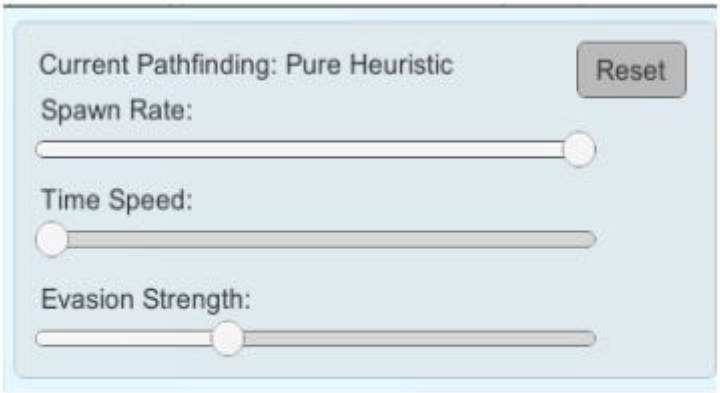


Figure 2.13 Newcastle Simulation Options

In this scene the GridManager cycles between updating the enter-diffusion and exit-diffusion value of each node, i.e. the first update computes enter-diffusion, the second exit-diffusion, the third enter-diffusion again, and so on. Entering agents follow the enter-diffusion value to navigate to a parking lot, while the existing ones follow the exit-diffusion to exit the city. This is done to optimise the frame rate of the simulation.

The algorithms

The pathfinding framework implements three algorithms: collaborative diffusion, A* and Djikstra. A* and Djikstra's implementations are relatively standard, both utilising an open and closed list [33]. Both generate a path through the graph that the avatars follow, without interacting with changes to the environment or other agents. The node class is the main engine of the diffusion algorithm. At each update it applies the diffusion equation (Equation 2) to all its neighbours (by outgoing edge). In the implementation of CD the formula is adapted to:

Equation 7

$$u_{0,t+1} = \frac{(1-d)}{N \cdot s_e}\left(u_{0,t} + D\sum_{i=1}^{n}\left(u_{i,t} - u_{0,}\right)\right)$$

Where:

n = number of neighbouring agents used as input for the diffusion equation

$u_{0,t}$ = diffusion value of centre agent

$u_{i,t}$ = diffusion value of neighbour agent (i > 0)

D = diffusion coefficient [0..0.5]

d = rate of decay [0..1]

N = number of agents currently on the node

s_e = evasion strength

The rate of decay linearly decays the diffusion value of each node. This is needed so that the algorithm does not fill the whole grid with the constant diffusion value, which would render pathfinding inoperable. The number of agents and the evasion strength represent the interaction factor, which would be considered as competition using Table 2.1.

Chapter 3: Design and Implementation

3.1 Overview

This section outlines what was done and why. It begins with explaining how the project was planned; the specification is then laid out, followed by the tools used and the implementation.

3.2 Planning

The project development followed an agile methodology, with the central aim revolving around Goal 2: Newcastle Simulation and Goal 4: Evaluation. The pathfinding framework was first analysed and a sheet of specifications and milestones was produced. The feature development was done in separate sprints with defined goals. The implementations were then tested using the testing environment and planning of the next sprint began (Fig. 3.1). The resulting specifications and analysis are outlined in the Specification section of this chapter. The only exception to the development cycle was the theoretical development regarding augmenting the CD algorithm.

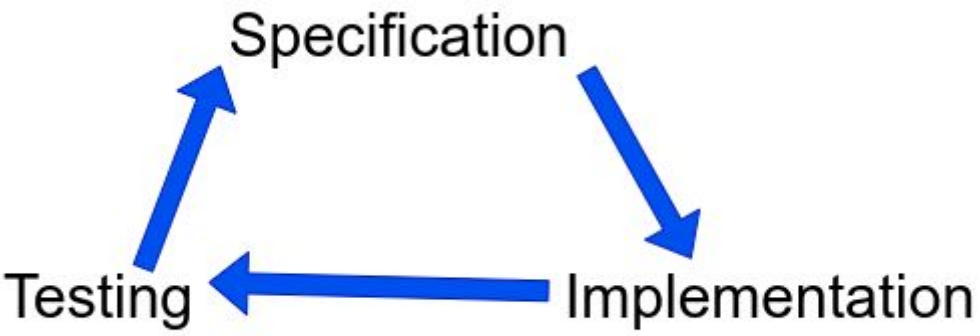


Figure 3.1 Cycle of Development

3.3 Specification

Before implementation of any kind could begin, the specification and scoping of the project had to be written. The main goal was to adapt the current project to a mid-fidelity PGI simulation of the Newcastle upon Tyne metropolitan area, employ CD and analyse the results. A mid-fidelity simulation is defined in this paper as one that is representative of the actual behaviours of different parts of a system (e.g. agents are able to collide with each other) without going into detail (e.g. ten vehicles can be represented by one agent).

The first step in the planning process was to understand and explore the previous system (see Background and Research: Pathfinding Framework). Following this step a critical assessment of the system was carried out with regards to the goals set out in this paper. From the research conducted (Background and Research: Smart Parking) a specification of what that system should represent was built using user stories. The format this takes is as follows:

“A (X) must (do/have Y) to (achieve aim Z)”

These stories were separated into three sections: simulation, evaluation tools and testing. Some of these user stories were part of the initial project planning, whilst others were subsequently added during the development cycle. They briefly encapsulate the abstract milestones of the project:

Simulation: the simulation of the PGI system must:

- Have a system for resource sharing to enable reservations;
- Have actors with specific goals to mimic real-life traffic and behaviours (i.e. commuters that go to specific places at specific times);
- Have control over induced traffic to better simulate real-world scenarios;
- Have a degree of realism in physical interactions and road networks to better simulate congestion;
- Have both collaborative and non-collaborative pathfinding, to be compared and evaluated.

Evaluation tools: the simulation should be able to extract data about:

- Traffic to be used as an indicator of how many cars are driving;
- Congestion to be used as an indicator of the results of the algorithms;
- Visually representing this traffic and congestion to enable easy analysis.

Testing: a testing environment for the simulation must:

- Have automated tests to save time and verify the correctness of tasks;
- Extract data from the algorithms into correct formats to further investigate behaviours;
- Visualise algorithms to help with their understanding.

Using the user stories and the general aim of the dissertation, the pathfinding framework was evaluated and concrete milestones created.

The simulation

The Newcastle simulation was found to be lacking in many respects with regard to the specification above. It was essentially the same as the testing environment, but on a Newcastle upon Tyne map, with uniformly random spawning, parking and control over agents' speeds. A* and Djikstra were also not implemented.

The first critique is that the mechanism of randomly spawned avatars could not be used to model the typical behaviour of traffic. Figure 3.2 shows the normal daily vehicle count in the Newcastle City Centre area (the area represented in the simulation) extracted from Urban Observatory's API [11]. It clearly shows the differences in traffic at different times of the day; thus, traffic is not entirely random, but instead closely relates to the hour of the day.

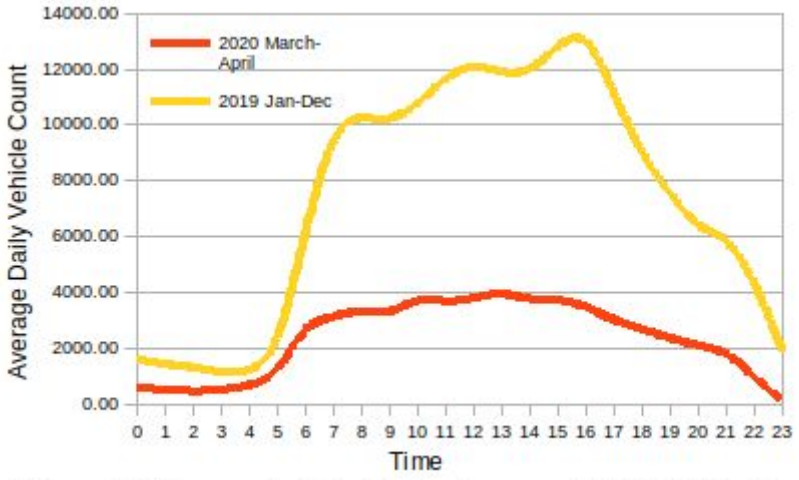


Figure 3.2 Newcastle City Centre Average Daily Vehicle Count

Another problem with the simulation is that it does not model the passing of time in any way. This is a major issue, especially since most of the data looked at is time-series data. The vehicle velocities and parking times are also not representative of real-life traffic.

The system does not include PGI functionalities for reservation and resource sharing. Agents do not have any goal or direction to even enable these behaviours; they only aim at the closest open parking space. The agents also do not interact with other agents, the city and the roads. This is problematic since congestion cannot thereby be measured in a meaningful way.

The evaluation tools

No evaluation tools were implemented in the pathfinding framework with regards to the specification. The only data displayed was an approximation of the revenue generated by parking.

The testing environment

The testing environment is mainly designed to compare and visualise results between pathfinding solutions. This scene generally presents a well-designed testing environment that fulfils the majority of the requirements. It visualises the diffusion algorithm by gradually changing the saturation using diffusion values (as previously seen in Fig. 2.6). It allows for custom scenario creation and spawning of avatars (Fig. 2.8 and Fig. 2.9). This can be used to manually set up environments and develop tests.

The only further developments that this scene required to meet the specification above was to automate the creation of maps, spawning of avatars and implementing a way of extracting data.

Milestones

From the above assessment, the following concrete milestones were developed for the project:

For simulation implementation

1. Non-collaborative solutions to compare with collaborative diffusion.
2. A clock and scaling all time related processes to the speed of that clock (i.e. speed of vehicles).
3. More realistic spawning that follows a normal distribution rather than a uniform distribution.
4. A way of injecting real-world traffic data into the simulation.
5. More realistic agent behaviour with regards to the aims of agents (i.e. they should have a goal parking lot or area).
6. Parking space reservation system.
7. Simple collisions between avatars and more realistic roads.

For evaluation tool implementation

1. General traffic statistics extraction: parking occupancy statistics, road occupancy statistics, agent path and travel time statistics.
2. Heatmap of traffic.

For testing implementation

1. Automated testing of scenarios.

3.4 Tools and Technologies

Unity and C#

The framework used for the development of the project was the Unity 3D game engine. The decision was made to continue development using the same platform as the original pathfinding framework.

A significant benefit of the Unity engine is that it allows for the quick scaffolding of ideas through an interactive interface. The engine offers a primary scripting API in C# and the option of writing native plugins for specific platforms. Cross-compilation for different platforms, including WebGL, is also a key benefit.

3.5 Design and Implementation

3.5.1 Simulation

Non-collaborative pathfinding

The implementation of A* pathfinding was added in the simulation to meet the specification for a non-collaborative algorithm. The implementation utilises a typical open/closed list. The inclusion of multiple goals and caching marked a departure from the normal algorithm.

The implementation consists of creating a new class, AStarGenerator, which is queried by agents to find the shortest path from their current position to a goal. Avatars save this path

and follow it until they either reach it or it becomes obstructed in some way (e.g. the parking lot closes). The original class structure (Fig. 2.7) is extended, as shown in Figure 3.3.

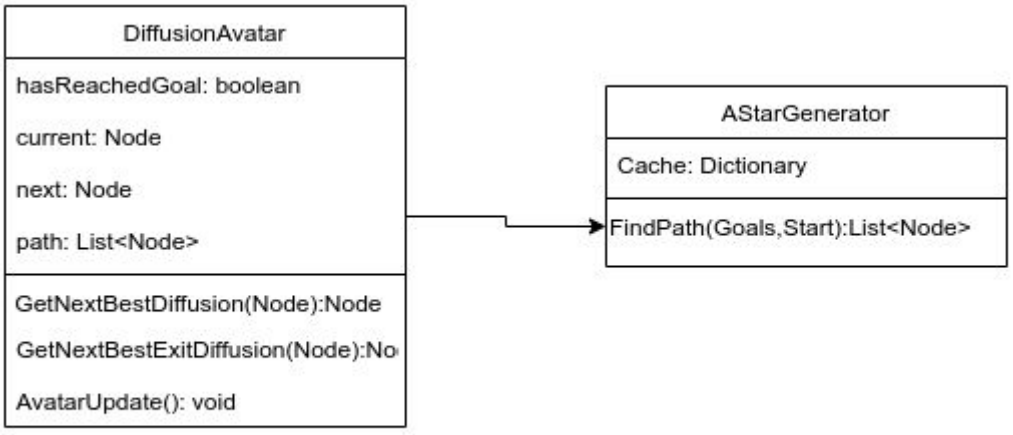


Figure 3.3 AStarGenerator Extension Class Diagram

Since a large number of agents simultaneously plot paths, A* causes large drops in frame rates. To avoid this, paths are cached. As avatars mostly navigate from/to the same positions, the paths can easily be saved and reused. Furthermore, for any particular goal and path, all nodes from that path already “know” the path towards the goal, since they are part of it. Figure 3.4 visualises this property.

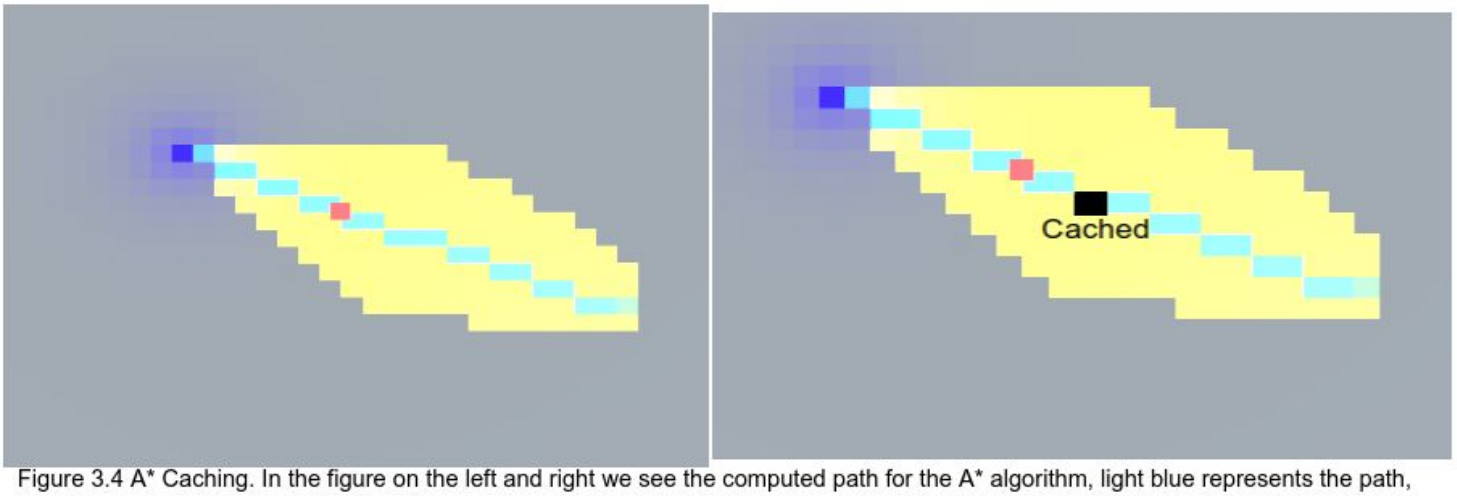


Figure 3.4 A* Caching. In the figure on the left and right we see the computed path for the A* algorithm, light blue represents the path, yellow - the searched nodes, red - the agent and purple - the goal.
To illustrate how the caching works, if one were to ask what is the shortest path from the position marked as cached(Figure on the right) our algorithm should not need to recalculate the shortest path from that position for that goal.
While A* is searching for paths it should check if the searched node already has a cached path

Using this property, each generated path is cached and the shortest path problem for the whole graph is iteratively solved. The implementation of this cache is a dictionary that points to a path for a given goal and current position. The structure is shown in Figure 3.5. Using this dictionary, the AStarGenerator class checks the searched nodes for already existing paths and stores previous results.

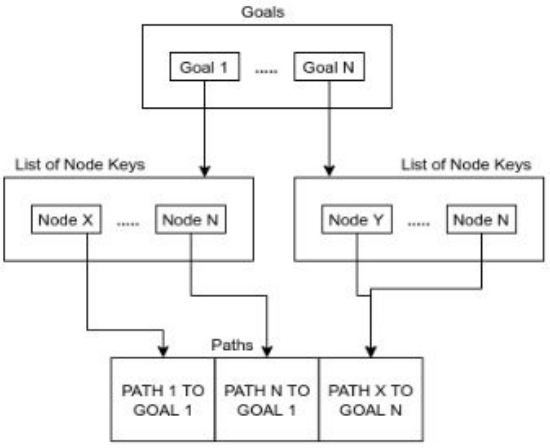


Figure 3.5 A* Cache Data Structure.
In the first level the goal represents the goal/s of an agent
In the second level the list of Node Keys are the already cached positions for a particular goal.
i.e. the shortest path is known for all nodes in the list for that specific goal
Each of these nodes points to a the shortest path to that specific goal.
They could point to the same path as well as seen in the left subbranch

The second adaptation to A* was to calculate the shortest path for multiple goals. Instead of calculating the heuristic value (using Pythagoras distance) for one node:

$$H = Pythagoras(\ goal,\ current)$$

It is calculated for all potential goals (defined by an agent) and the minimum value is taken:

$$H = MIN(\ \ Pythagoras(\ goals,\ current)\ \ )$$ *(assuming Pythagoras (goals, current) returns a list of distances)*

Redefining the heuristic as such allows the algorithm to find the shortest path to the closest goal.

These optimisations allow the simulation environment to run at an acceptable frame rate while using A*.

Non-collaborative diffusion

To better understand and experiment with the collaborative diffusion algorithm, a simple implementation of “non-collaborative” diffusion is added. This is later used to evaluate whether agent interaction is responsible for traffic alleviation. The only difference between this algorithm and Equation 7 is that the agent interaction term is removed. The original algorithm is transformed from:

$$u_{0,t+1} = \frac{(1-d)}{N \cdot s_e}\left(u_{0,t} + D\sum_{i=1}^{n}\left(u_{i,t} - u_{0,}\right)\right)$$ *(eq.7)*

to:

Equation 8

$$u_{0,t+1} = (1-d)\left(u_{0,t} + D\sum_{i=1}^{n}\left(u_{i,t} - u_{0,}\right)\right)$$

Time

To simulate the passing of time, a “clock” is created. This is done by continuously calculating the time between frames (delta time), which equates to the passed time from the start of the simulation up to the current moment. The value is then used to calculate the seconds, minutes and hours of the day in our simulation. Delta time is also scaled (by seconds per second) to allow for control of the “speed” of time:

Equation 9

$$seconds \; += \; Time.deltaTime \cdot secondsPerSecond$$

$$minutes = \frac{seconds}{60}$$

$$hours = \frac{minutes}{60}$$

The clock is presented in a simple UI and the previous UI slider for speed now controls the seconds per second term (Fig. 3.6).

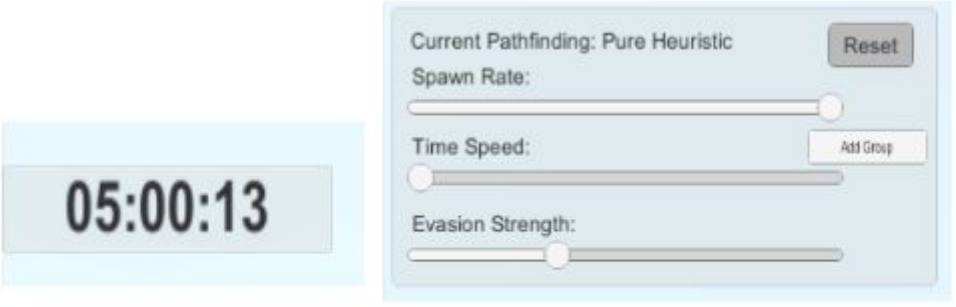


Figure 3.5 Clock UI and Control Slider

The speed of the passing of time is used to adjust all time-related processes and especially the speed of vehicles. To calculate vehicle velocities, the scale of the map needs to be accounted for. It was found that one mile accounts for 240 simulation units. To calculate the velocity, a constant speed is given to each agent (in miles per hour). This value is transformed from miles per hour to game miles per second. Then, using delta time and the time speed up, the distance covered between frames is calculated using Equation 10. This distance is then applied to the next node of the agent.

Equation 10

$$from$$

$$S = v \cdot t$$

$$we\ have:$$

$$PixelDistancePerFrame = \frac{(speed \cdot 240)}{3600} \cdot secondsPerSecond \cdot deltaTime$$

Groups and spawning

To achieve the set milestone for realistically distributed spawn times and agent goals, the group class is introduced to the project structure. The idea of the group class is to allow for multiple groups of agents with different spawn times and goals to be managed in a distributed and sensible manner. Adding and removing groups occurs through the UI interface (Fig. 3.6).

Fig 3.6 Groups Interface. The bottom left dropdown menu selects the goal The percentage capacity and the radius are not editable from the user interface.

A group has three main responsibilities: managing spawning and parking time, directing agents, and setting goal areas.

Spawning

The most important aspect of groups is controlling spawning and parking for agents. This allows for the simulation of a more accurate flow of traffic, rather than just randomly spawning agents. The spawning behaviour should be normally distributed around a specific hour of the day (rush hour).

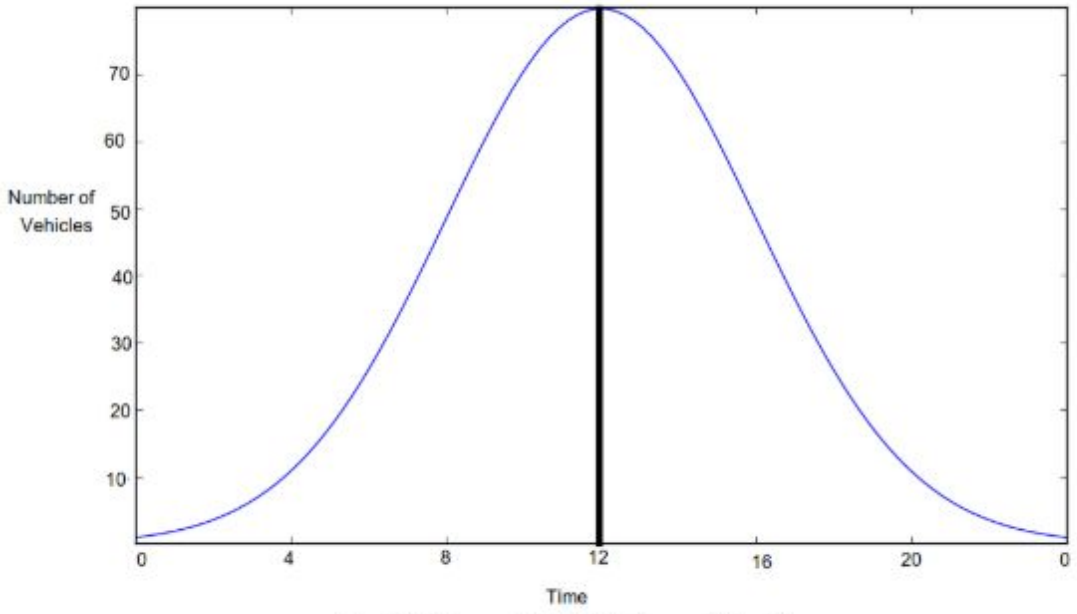


Fig 3.7 Normal Distribution of Traffic

Figure 3.7 visualises this idea. The x-axis is the time of the day and the y-axis is the number of vehicles spawned. The “rush hour” of 12am is represented as the mean of the distribution, that is, most of the vehicles will have been spawned around 12am. The “shoulders” (standard deviation) is represented as hours. In the chart, one standard distribution would be four hours. Finally, the height of the distribution is determined by the overall number of vehicles spawned.

The implementation of this distribution entails two parts: distributing the spawn times and spawning the agents. To achieve the former, the group class contains attributes that control each part of the distribution (i.e. a variable for the mean, standard deviation and number of vehicles to spawn). It also contains a parking time variable that controls how much time agents from this group wait before they leave a parking lot. The parking time is set to a specific value rather than randomised because adding multiple distributions together would distort the simple modelling.

Using the mean and standard deviation, the group samples a normal distribution (with the MathNet library) for each vehicle it has to spawn. The resulting values are then put into “spawn buckets” for each minute of the day. These are represented by a map of the time of the day to number of vehicles to spawn. For example, if 30 avatars were spawned, with a mean of 4 and a standard deviation of 1 hour, it would be expected that the following number of vehicles would be created to spawn each hour (Fig 3.8):

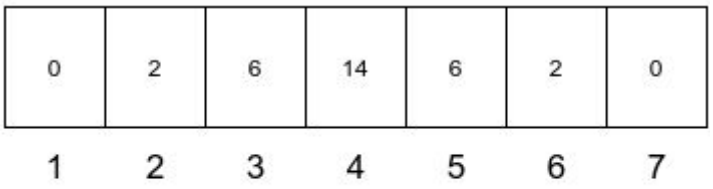


Fig 3.8 Spawner Bucket Example
Time is the numbers below the boxes
The number inside the boxes are agents to spawn

Generated spawn buckets are then equally distributed among all spawner nodes. Note that a spawner node contains multiple buckets for each group. At each update the node class accesses the current time and checks how many agents to spawn at that particular time. The agent contains information about its group so that it can track the goal of that group and park for the specified time.

The direct checking of time is vulnerable to frame drops, however, where checks for particular hours might be lost. To avoid this, nodes continuously compare how many agents have been spawned and how many should have been spawned. The difference is then rolled over to the current spawn bucket shown in Figure 3.9.

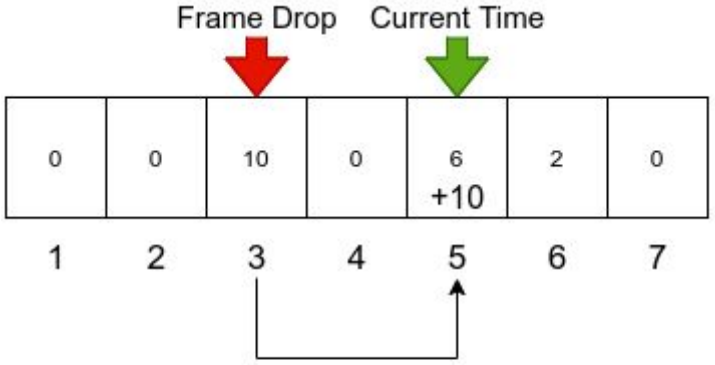


Fig 3.9 Frame Drop Example,
Since time-step 3 has been skipped
due to frame drops, the agents to spawn
roll over to the current time step

Stepping on this implementation of spawning data, injection is also employed. The idea is to allow users to inject real time-series traffic data into the simulation. The “spawn bucket” mechanism allows for easy integration with csv data. The “time” and “value” fields of this csv data are parsed into spawn buckets and then passed to the nodes (instead of the randomised data).

To enable more customisation of this injection, behaviours can also be specified through another csv file (Table 3.1). The fields of this file are as follows:

- *groupName*: this is the name of the group that will be created.

- *startTime and endTime*: the time in these two fields is the time of day that encapsulates the group (i.e. any agents after startTime but before endTime will be part of this group). If they are not set, the program will assume that this entry is the default behaviour for all agents not belonging to a group.

- *parkingTime*: the time that agents within this group will park.

- *parkingGoalName*: the parking goal name is the name of a specific parking lot in the simulation. This is used to direct avatars towards a specific area (or none).

Table 3.1 Behaviour settings

groupName	startTime	endTime	parkingTime (hours)	parkingGoalName
Work Commuters	6	9	8	The Sage
Shoppers	10	13	2	None

The mechanisms above allow us to build versatile simulations and explore different scenarios.

Goal direction

Directing agents towards areas or a specific goal for A* is just a matter of passing a specific goal rather than an array of goals. However, for collaborative diffusion a more involved solution is required. This problem was also faced in the original pathfinding framework, where agents were programmed to both enter and leave the city. The solution was to “intertwine” different diffusion values that did not interact with each other, one for entering and one for exiting. Each node held two diffusion values, as shown in Figure 3.10. This allowed avatars to follow only one of them, but still interact by decreasing both values on nodes they have passed.

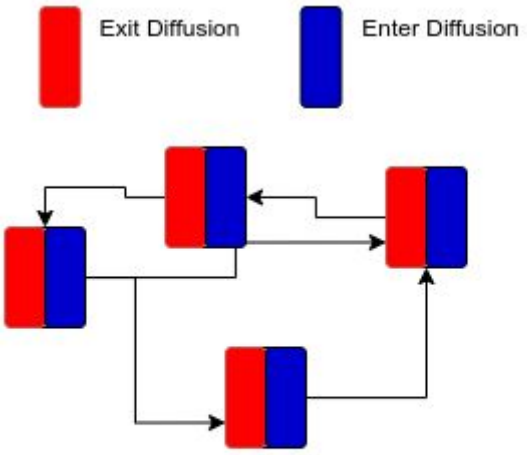


Fig 3.10 Agent Diffusion values.
Each node holds and updates 2 values
exit and enter diffusion

To direct agents to specific goals, the same approach is adopted and generalised for multiple goals. To implement this behaviour, the node class is adapted to hold a dictionary with a key of type group and a corresponding diffusion value. The enter and exit diffusion are kept and used for agents that do not have a specific goal. These diffusion values are updated separately from each other. The result of this is that Figure 3.10 becomes Figure 3.11.

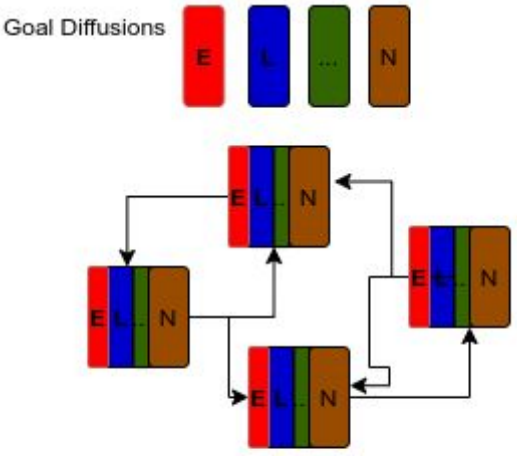


Fig 3.11 Agent Diffusion implementation
Each node holds and updates N-diffusion
values

Note: This implementation only allows vehicles to go to specific parking space goals. The implementation could be adapted so as to store both an enter and leave diffusion for each group, but the current solution was deemed sufficient to simulate driver behaviour.

In Figure 3.11 each node holds N amount of diffusion values. Agents use these values to navigate towards a goal. Whenever a new group is created an entry is added to the diffusion dictionary of each node class and, if it is the goal, its diffusion value is kept constant. The

agents spawned under this group then follow that diffusion value only, which leads them to the group goal.

From this implementation, another very similar problem to one in the original project emerged. Multiple diffusion values for separate goals caused a large decrease in frame rate. The original behaviour of the updates is represented in Figure 3.12.

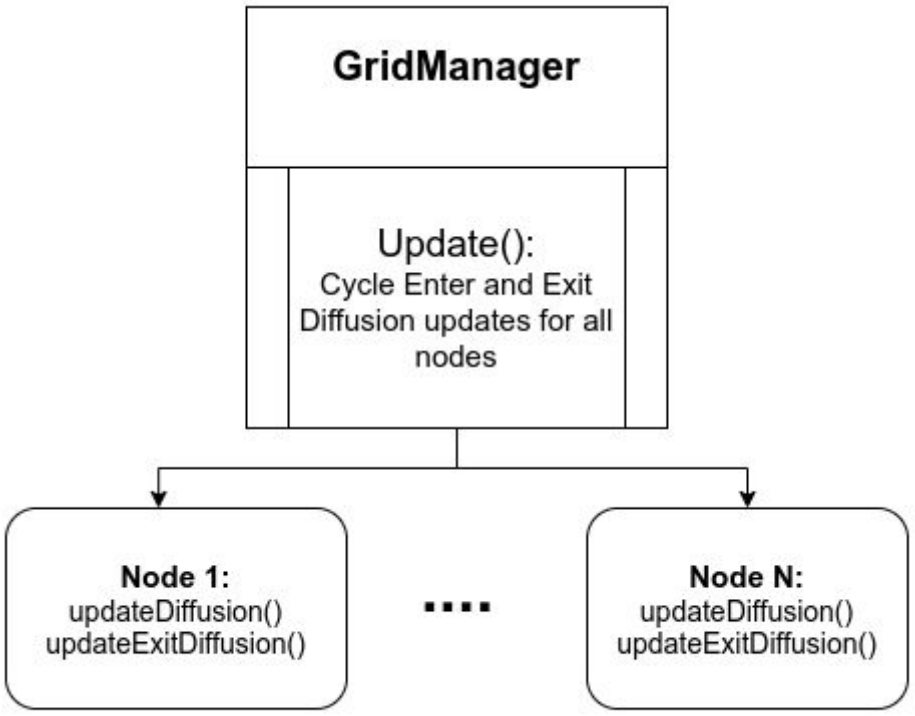


Fig 3.12 Diffusion Update Schema

Here, each node received an update that was directly scheduled and controlled by the GridManager. The manager here only has to keep track of two diffusion values, one for entering and one for exiting. When multiple goals are introduced, each node has to update the diffusion value for each of the goals. If there were 10 nodes and 10 groups with different goals, the program would be waiting for 100 diffusion updates. This means that the GridManager update function (which runs every frame) has a cost of *O(n)* and grows linearly with the increase of goals. This is a problem since this function is run every frame. To keep it O(1) (constant), updates between each of the goal nodes are cycled and the solution above is generalised to work for multiple groups (Fig. 3.13).

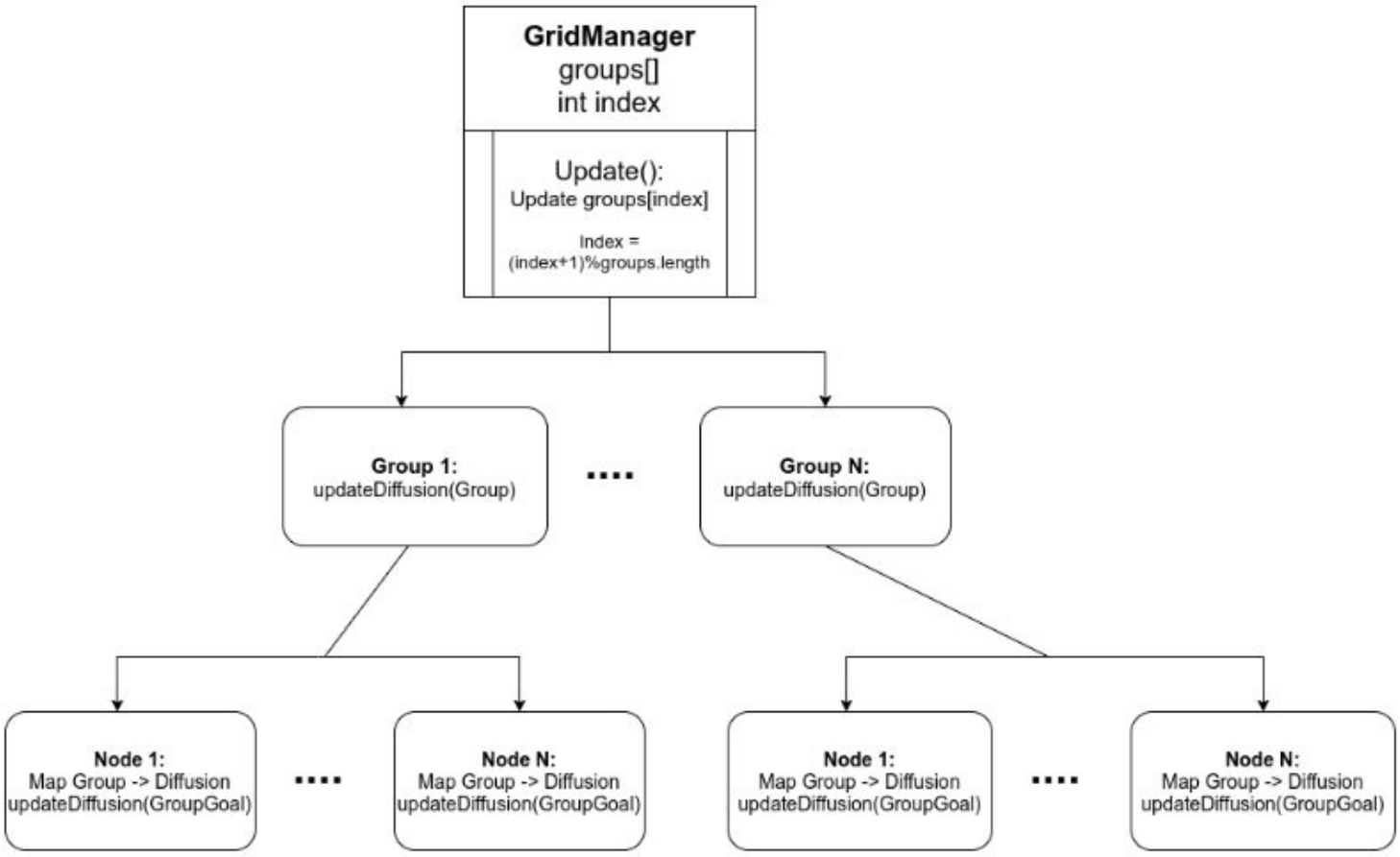


Fig 3.13 Updated Diffusion Update Schema

Figure 3.13 shows the new implementation of updating diffusion. The GridManager cycles updates for each of the groups. On updating, a group calls for the calculation of a particular diffusion value within each node. The node stores a map of a group (key) pointing towards a diffusion value. Agents follow the diffusion for their particular group only. Making updates in this fashion allows us to keep the cost of the GridManager updates constant. This is a useful approach since the GridManager already knows about all created groups, but it does not know about the goal of a particular group. The only problem is that it breaks some OOP concepts since the list of nodes on the grid has to be made public and static because it is unknown to a particular group.

Goal areas

The last group mechanism to implement is the setting of goal areas. Groups are meant to direct agents to regions rather than specific goals, to represent realistic behaviour. For example, if a commuter's goal was to go to work at the Urban Sciences building, he or she would most likely prefer to park in the parking lot for that building, but could also park

somewhere nearby, especially if the parking lot were full. To mimic this behaviour, the goal regions in the simulation have to be dynamic, that is, they should grow when a parking lot fills and shrink when it is empty again.

The implementation of this is straightforward. Each group is created with one goal: its origin goal. Anything within the reach of a specified radius from that origin will be added to an array of goal nodes (as seen in Fig. 3.14).

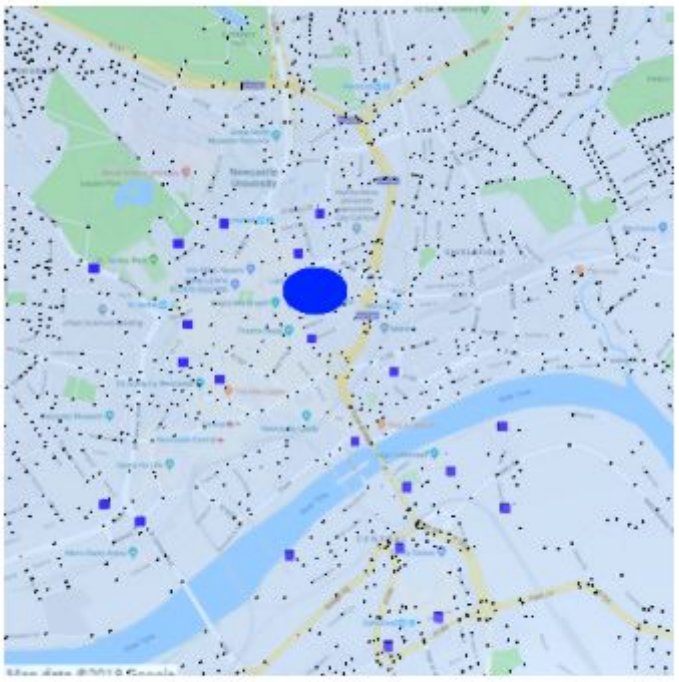


Fig 3.14 Initial Goal Radius

On each update, the group class checks if the parking spaces from all current goals (including the origin) are over 70% capacity. If so, the original radius expands to the next nearest parking from the outermost current goal. All parking lots within that radius are then added to the goal array, as in Figure 3.15.

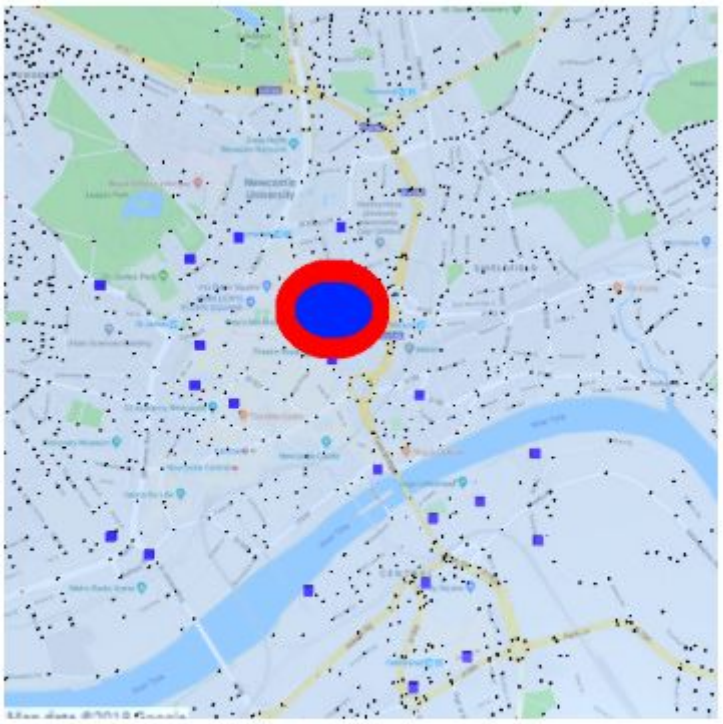


Fig 3.15 Expanded Goal Radius
(Expansion is in red)

If the goals from the inner region move below 70% capacity, this region will shrink back to Figure 3.14. This continuous expansion and shrinkage creates a dynamic region that the agents from a particular group are directed towards.

Group summary

The resulting interface from this can be seen in Figure 3.6. The group class fulfils the following three requirements: realistic spawning, injecting real-world data, and realistic agent behaviour in terms of goal and direction. A summary of how the group class extends the current class structure can be seen in Figure 3.16.

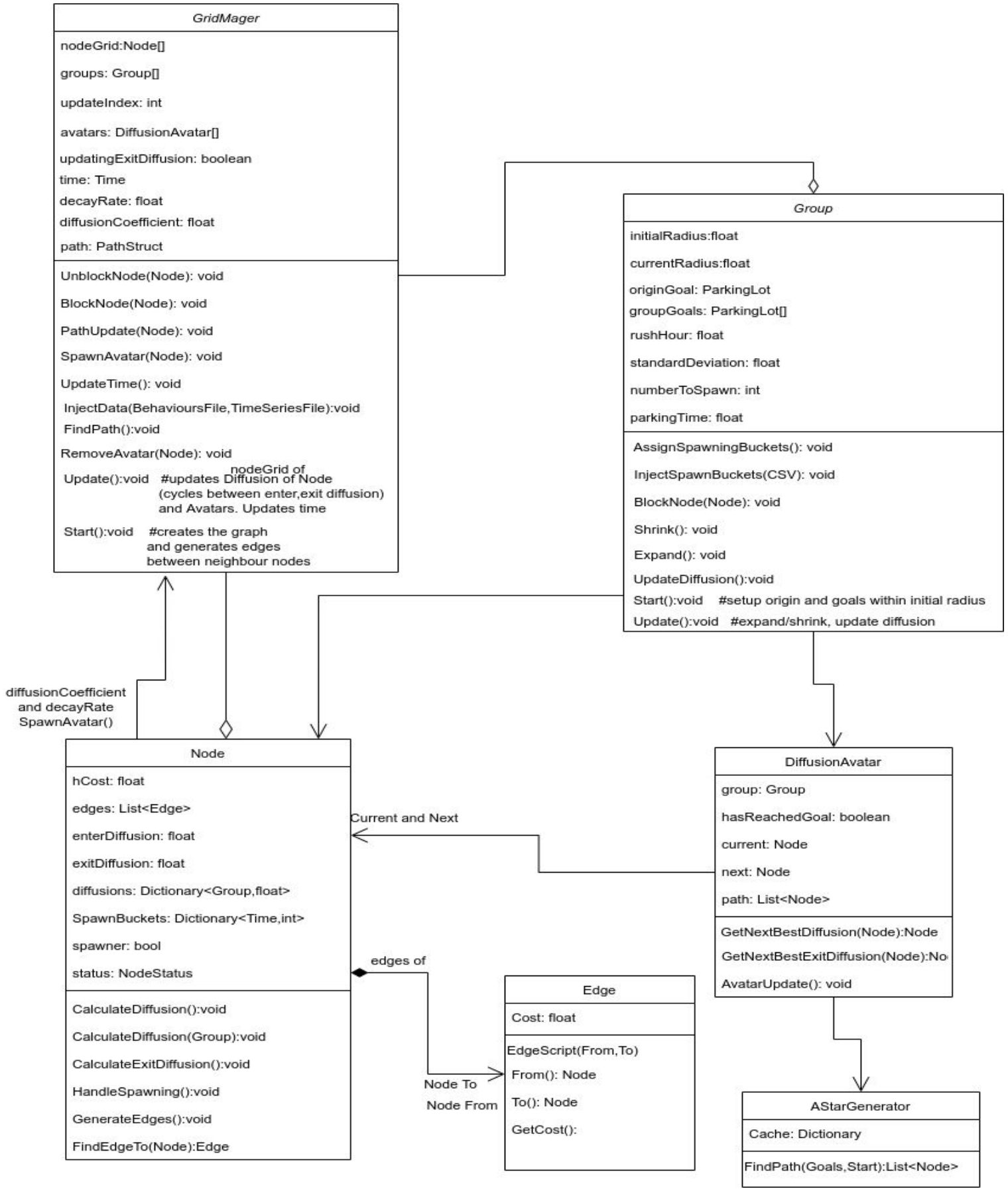


Fig. 3.16 Class diagram – groups

Realism

As described in the specification, the road systems and agent interactions are not “realistic” enough for a meso-scale simulation. To achieve the milestone of realism in agent interactions, collisions are added into the simulation. For more realistic road networks, traffic lights, street parking, road interactions and road congestion are implemented.

Collisions

As described in the specification brief, some level of physical interaction between agents is required. Specifically in the simulation, agents are programmed to mimic “stop-and-go” events. “Stop-and-go” behaviour in real life terms is characterised by periodically enforced stops, caused by heavy traffic. In the simulations, agents imitate this behaviour by stopping each other upon collision. When an agent vehicle “bumps” into another agent vehicle from behind, the former stops until the latter is no longer colliding with it, as seen in Figure 3.17.

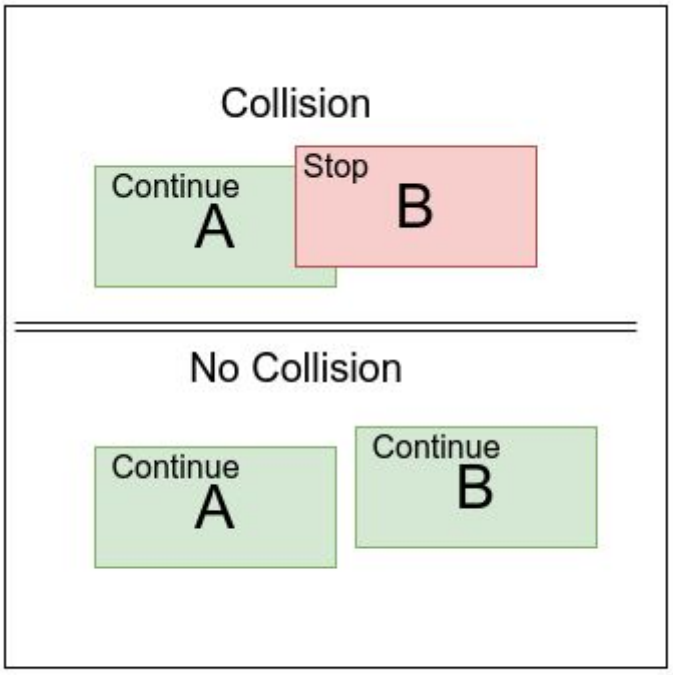


Fig 3.17 Stop and Go

The implementation includes adding a trigger class that manages collisions using Unity native collision mechanics. This is directly linked to the DiffusionAvatar instance. An additional condition for a collision to occur is that both agents have to be headed towards the same node. This allows agents to travel in different lanes without colliding.

Upon collision, an agent checks if the colliding agent (other) is in front or behind it by comparing the distance between the two objects and the node they are travelling to, as seen in Figure 3.18. Nothing happens to the agent at the front, while the other agent triggers a signal that it is supposed to stop.

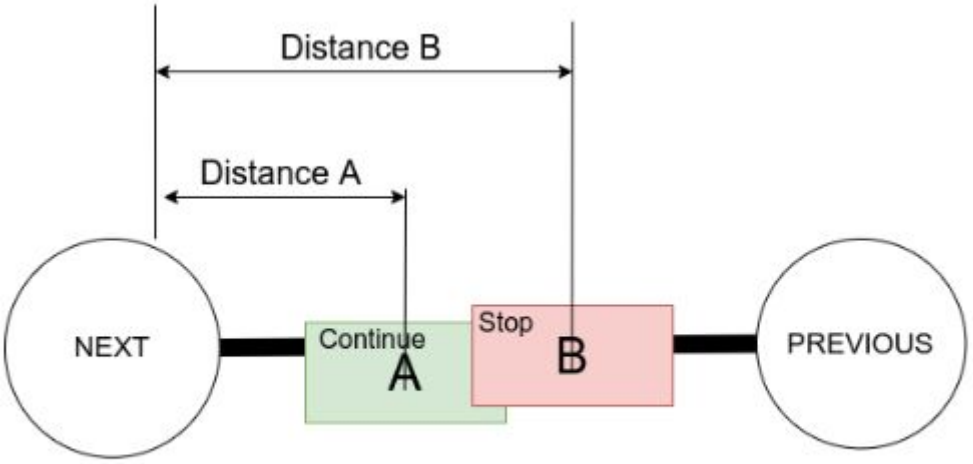


Fig 3.18 Distance Calculations

Upon exiting the collision, the stopped agent's stop flag is removed. This minor implementation allows agents to interact on a much deeper level and to create measurable amounts of traffic. The adaptation of the class structure can be seen in Figure 3.19.

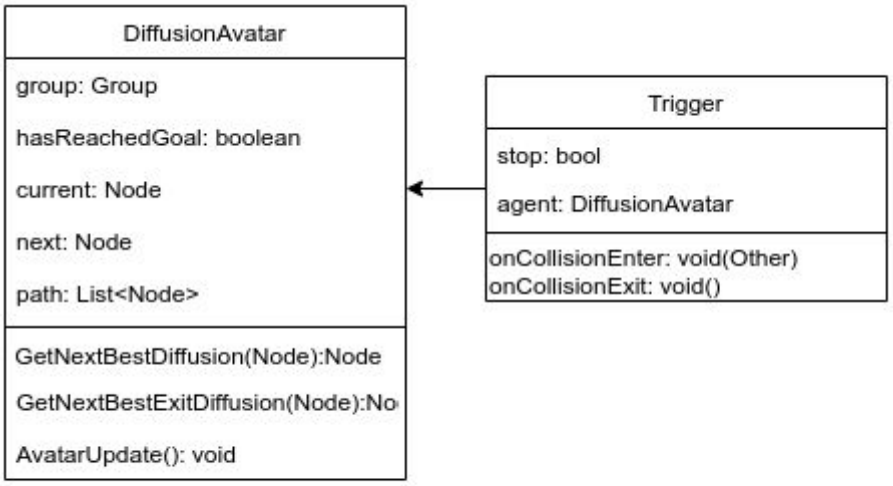


Fig 3.19 Diffusion Avatar Extension

Speed limits and roads

To create a more realistic road system, different speeds are defined for different roads. In the original implementation, speed was a constant for all agents. A speed limit essentially only changes that constant. The limitation of roads is an important factor in commuter decision making and limitations. Drivers generally prefer to travel on a highway or bigger road rather than small city roads. Another aspect of roads is that only a certain number of agents can travel on them at the same time before congestion ensues.

To replicate speed limits in the simulation, nodes will be flagged as either a normal road or a highway road (as seen in Fig. 3.20). Agents travelling on these roads will have different speeds, which can be specified in the UI interface (Fig. 3.21).

Fig 3.20 Speed Control UI

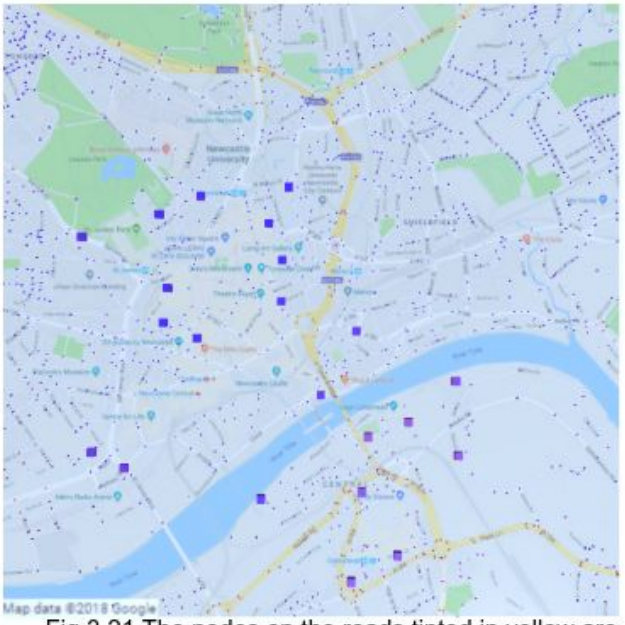


Fig 3.21 The nodes on the roads tinted in yellow are marked as highway nodes, the rest are city nodes

Upon the creation of edges between nodes, the edge class checks the status of the nodes it is connecting and saves a speed value for that edge. Now, instead of using a constant value for speed, agents use the speed value of the edge that connects its previous and next nodes.

To mimic road congestion, each edge is also assigned a capacity. This is done by calculating the distance between two nodes and dividing it by the size of the agents. As the occupancy of a particular edge increases (i.e. more agents travel between the nodes they are connecting), its speed limit value decreases (e.g. Fig. 3.22). This decrease is capped at 30% of the original speed limit.

Running at 30% capacity

EMPTY	AGENT	AGENT	AGENT	AGENT

Fig 3.22 Road Example.
Each square represents a capacity slot of the road.

Street parking is also considered as a part of realistic road interactions. This is an important part of any traffic simulation, since they play a big part in where and how people park [2]. A street parking spot is created during the edge creation between neighbouring non-highway nodes. On a random basis (controlled by the GridManager), an additional node is spawned between two neighbouring nodes (Fig. 3.23). It is marked as a normal parking lot, but its capacity is set to the capacity of the road it is on.

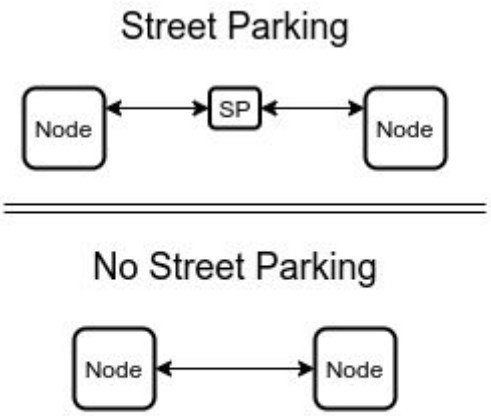


Fig 3.23 Street Parking
SP- is a small parking spot

Traffic lights

Traffic lights are one of the main contributors to “stop-and-go” vehicle behaviours, therefore they are a vital part of the road network. To find the locations of all traffic lights in Newcastle City Centre, OSM’s Overpass Turbo API [47] was used. The positions of all traffic lights placed in the simulation can be seen in Figure 3.24.

Fig 3.24 Traffic light positions
A crossed out circle means that a traffic light is placed at that postion

The implementation of street lights uses the same “stop-and-go” mechanism employed for vehicle collisions. A traffic light class is attached to the same Unity GameObject of a node class. Every two minutes (scaled by simulation time) the traffic light changes colour. Whenever a traffic light is red, it is marked as a colliding agent for all agents passing it. This stops any agent from colliding with that node until the lights turn green. The changing of traffic lights and agent collision allows for “traffic jams” to occur within the simulation (Fig. 3.25).

Fig 3.25 Traffic Jam.
The leaving avatars are lined up and waiting for a green light

Reservations

The final milestone is the reservation system. Here, an attempt is made to simulate the behaviour of a PGI system making reservations. In the Newcastle simulation, groups allow a percentage of their agents to reserve parking spaces at chosen locations.

Reservations work as follows:

1. A spawned agent is chosen to reserve a parking space.
2. The parking lot reserves that space and saves the reference of the agent who made the reservation.
3. When the agent arrives, the agent is parked and all reservation information is deleted.

To enable this to work, parking lots hold a list of agents that have reservations. The overall capacity of the parking lot is decreased by the length of the list (i.e. each reservation removes a capacity slot). Upon spawning, a reserved agent adds itself to the list of its designated parking lot, as shown in Figure 3.26. This means that the space cannot be occupied by any other vehicle. When that agent arrives at its destination, it removes its reservation and parks for the designated time.

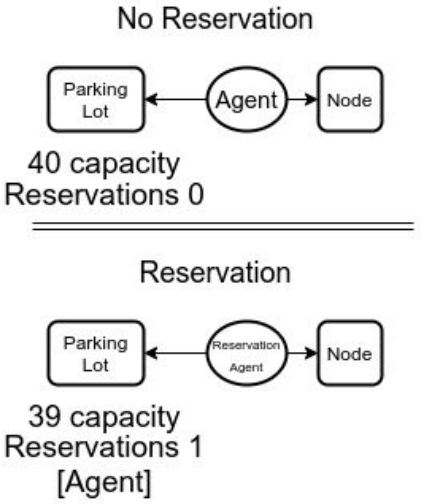


Fig 3.26 Reservation Vs No Reservation Behaviour

The implementation of the reservation system is generally straightforward, but the navigation of agents is slightly more complex. For A* the path is directly plotted and followed. However, to allow different goals for collaborative diffusion, separate diffusion values must be calculated for each goal. This presents a problem when the number of reservations increases. Potentially, there might be hundreds of different diffusion “scents” being calculated around the map if this approach is adopted. Due to this limitation of diffusion, agents navigate towards their goal using A* pathfinding only.

The controls for the reservation system are available as a slider to control the percentage of reservations made and a Dropbox to select the reserved parking lot (Fig. 3.27).

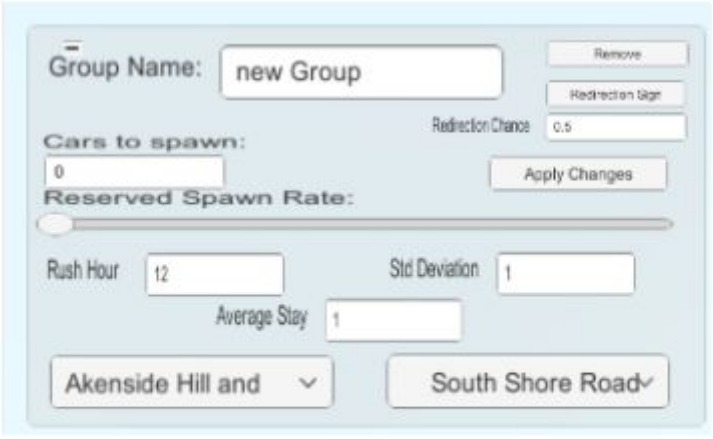


Fig 3.27 Reservation UI

3.5.2 Evaluation Tools

Data extraction

Data from the traffic needs to be captured and analysed in a meaningful way. Most of the data outputted by the simulation is in the form of time-series data because the generation of traffic (i.e. spawning of agents) is wholly reliant on the time of day. This data is captured by a dictionary that maps a time to the value collected. The data is then aggregated in different ways using C# lambda functions and it is saved as a csv file. Below is a comprehensive list of the data collected from a run of the simulation:

- Parking space occupancies over time: the occupancy of each parking lot over time, measured at each point when an agent enters or leaves.
- Flow for each node: the number of agents that have passed over a node.
- Flow over time: the flow for each node value is calculated and aggregated with respect to the time that the agents passed the node.
- Road occupancy over time: the aggregate occupancy of each road over time, measured at each point when an agent enters or leaves a given road.
- Agent time taken: the average time taken for an agent to arrive at a destination at any given time.

The groups used to generate this data can also be saved and loaded in the future.

Visualisation

To visualise data during the simulation, a heatmap is continuously computed using a shader. The resulting map is then laid over the Newcastle map, as seen in Figure 3.28.

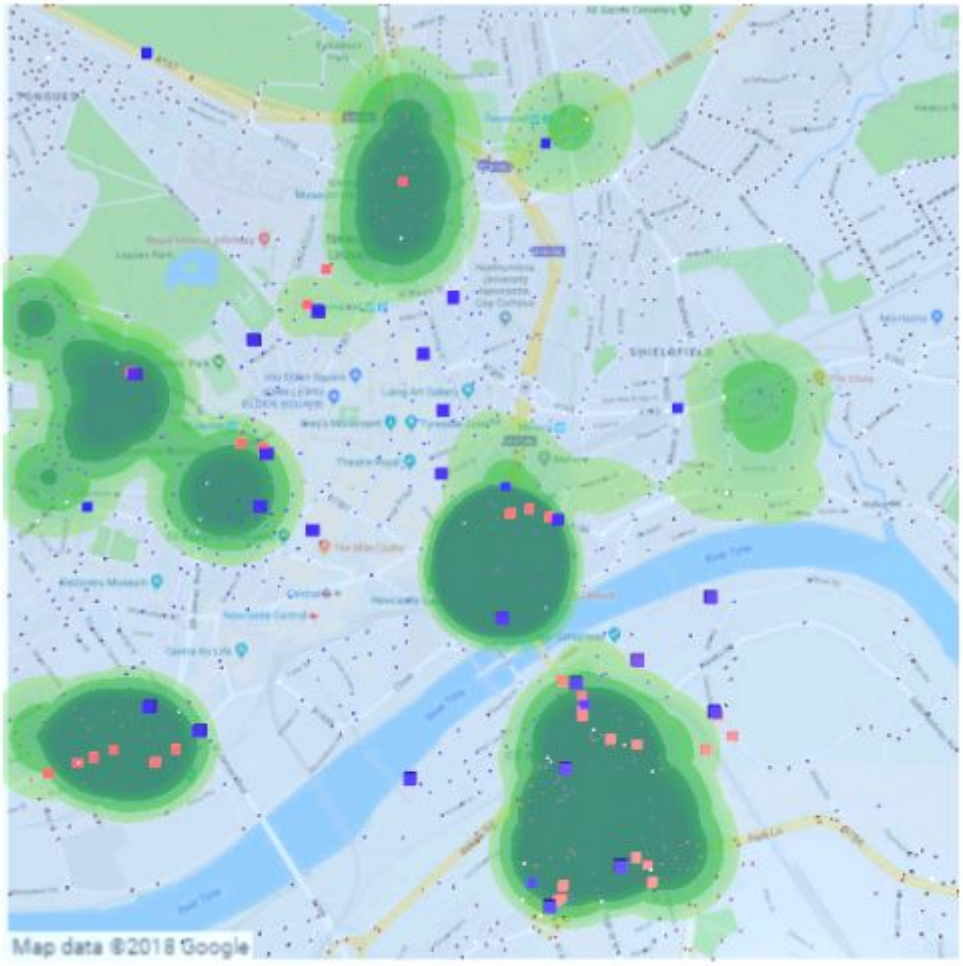


Fig 3.28 Heatmap generated by traffic

The heatmap generation works by:

1. Extracting data
2. Separating data into regions
3. Computing shader

To extract the data, the same sources described in data extraction are used. The current heatmap is hardcoded to only work with the number of agents passed (flow) at each node. This could be easily abstracted and the data passed could be, for example, road capacities.

Since the overall number of all nodes is too high, the nodes are clustered. This is done with the aim of reducing the computation cost of the heatmap shader. The map is separated into a variable number (specified in the configuration, always below the number of nodes) of square clusters (e.g. Fig. 3.29). The flow value for all nodes within a cluster is calculated. This sum is then used in the heatmap as the contribution of this region.

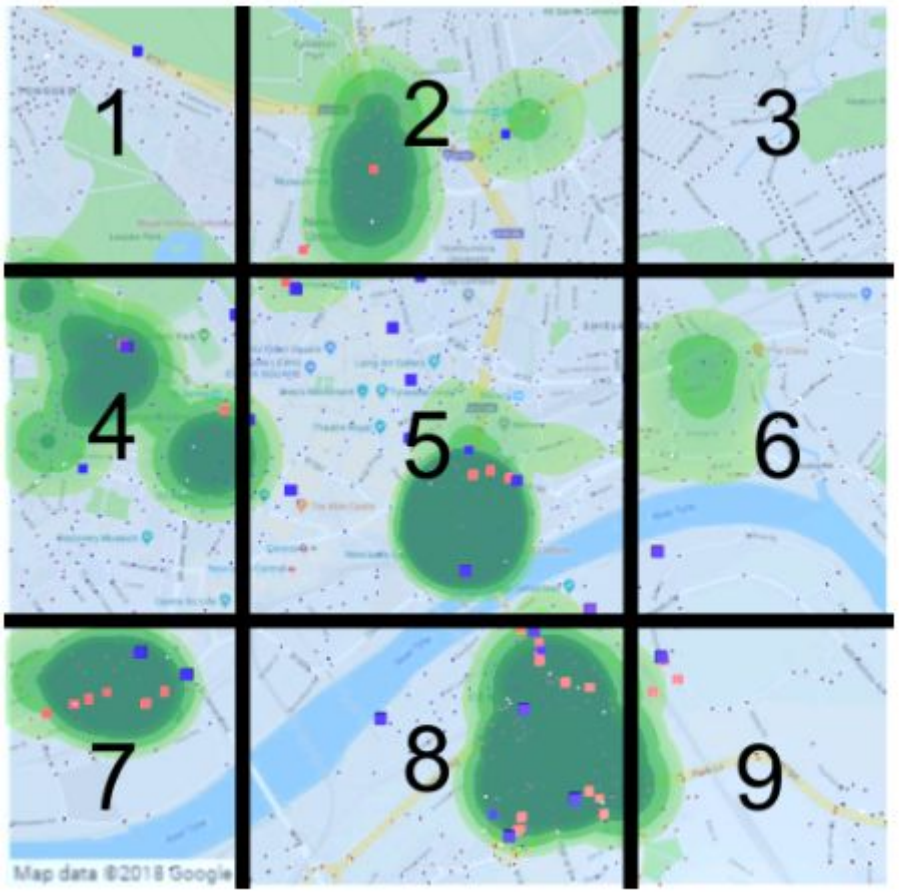


Fig 3.29 Heatmap Regions

The shader continuously aggregates and calculates the contribution of each region. Each contribution (relative to all contributions) is then used to saturate heatmap texture in the position of the region. This texture is then laid over the Newcastle map.

3.5.3 Testing Environment

Tests

Simple auto generated environments are added to the testing environment to satisfy the corresponding milestone. This does three things: it blocks nodes to create a scenario, places the start and end nodes, and continuously spawns avatars at a certain position, using a specified pathfinding algorithm. An example of a generated scenario can be seen in Figure 3.30.

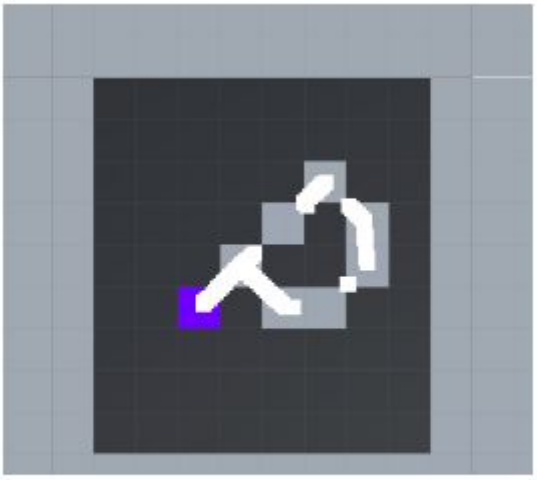


Fig 3.30 Testing Environment
Generated Scenario

Easily editable text files containing the positions and settings of all the nodes (and their type) are used to specify different scenarios. The test file is then passed to a generator script via the Unity editor. An example of such a file is available in Appendix 2.

Data about the grid itself can also be taken from the testing environment. More specifically, functionality for measuring diffusion and its effects on the field, including the average rate of change of diffusion at each node and the normalised diffusion values of all nodes, are ordered by distance from the goal.

3.5.4 Conformity Factor

Collaborative diffusion pathfinding augmented lim of dif

During the development of the project a problem was discovered with the collaborative diffusion pathfinding. Agents at this segment (Fig. 3.31) of the map were seen to be taking only one route to their goal. Further testing showed that the avatars did not take the other route even when the original was congested. They continued to follow the original route, which further built up congestion. Occasionally, agents were seen taking the other route, but the vast majority continued using the same path.

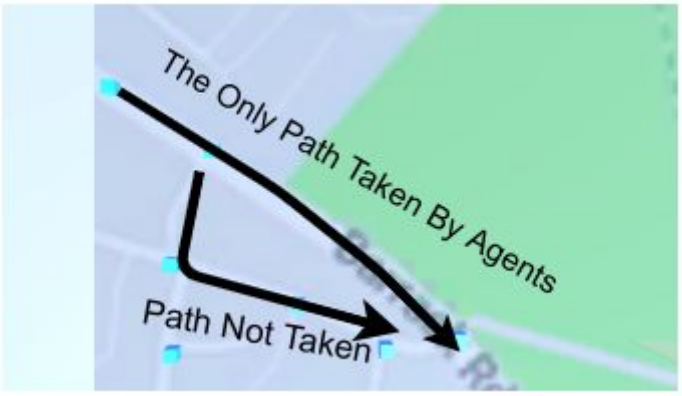


Fig. 3.31 Bottleneck problem

Upon further inspection, this seemed to be a problem in all places where:

1. The goal node dissipates its diffusion value to only one node, which further dissipates the constant value through itself to all other nodes. It can be said that node position two is acting as a “proxy” for the goal node (Fig. 3.32).
2. There is a closed system directed path (Fig. 3.32), rather than the open field (Fig. 3.33) seen in the testing environment.

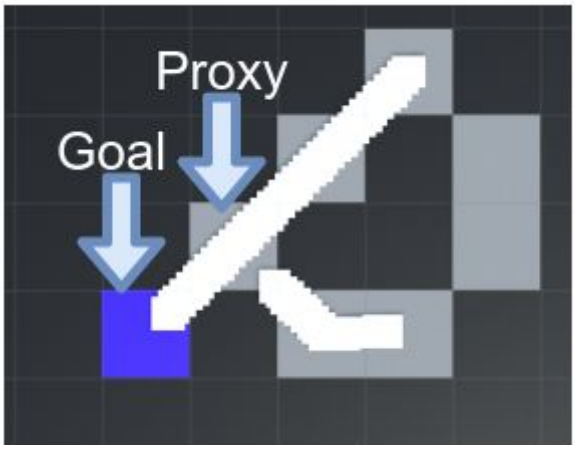


Fig. 3.32 Proxy closed system

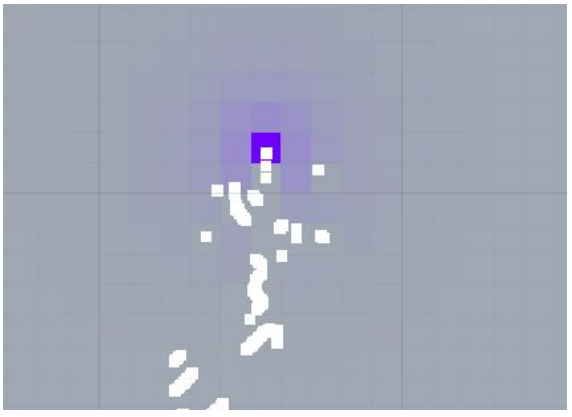


Fig. 3.33 Open field

This problem will be analysed in three ways – first, a trace is made and discussed, then the situation is simulated in the testing environment and further analysed. This section then ends with a formal evaluation and solution to the problem.

Trace

First, an environment similar to the one in the segment shown in Figure 3.31 is set up; this can be seen in Figure 3.34. The node at position 1 is the goal node and the node at position 4 is the spawner node. In this scenario there are two paths that an agent can take. Path A, which is shorter (2 nodes away from the goal), is marked in black, and Path B, which is longer (5 nodes away), is marked in blue. Here it is assumed that agents can travel diagonally between connected nodes.

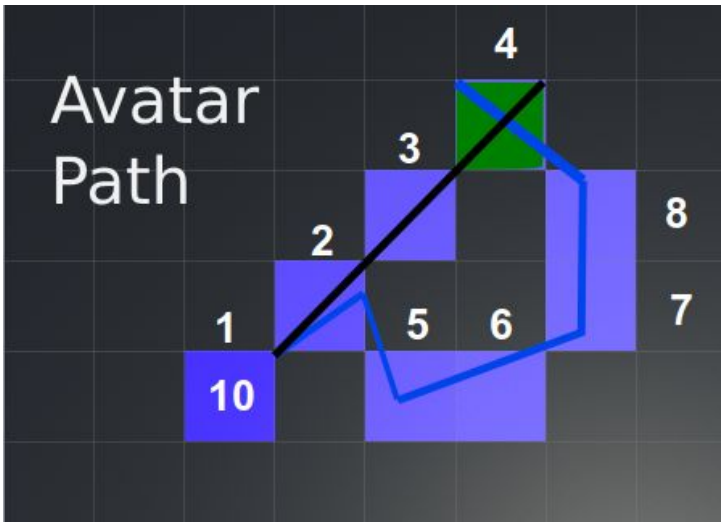


Fig. 3.34 Trace field

Field setup

First, the scenario with the field presented in Figure 3.34 is considered. As described in Chapter 2: Background and Research, the algorithm dissipates the diffusion value from the goal node to the other nodes. The following parameters for the collaborative diffusion algorithm are used (Eq. 7): diffusion coefficient *D*=0.4, decay rate d=0.1, evasion strength s_e =2, with the goal node with a constant value of 10. The values of each node at each time step are calculated with the following formulas (based on the agent-based diffusion formula described in Chapter 2: Background Research):

Equation 11

$$Node_{2,t+1} = \frac{(1-d)}{N \cdot s_e}(Node_{2,t} + D*(Node_{1,t} - Node_{2,t} + Node_{3,t} - Node_{2,t} + Node_{5,t} - Node_{2,t}))$$

$$Node_{3,t+1} = \frac{(1-d)}{N \cdot s_e}(Node_{3,t} + D(Node_{4,t} - Node_{3,t} + Node_{2,t} - Node_{3,t}))$$

$$Node_{4,t+1} = \frac{(1-d)}{N \cdot s_e}(Node_{4,t} + D(Node_{3,t} - Node_{4,t} + Node_{8,t} - Node_{4,t}))$$

$$Node_{5,t+1} = \frac{(1-d)}{N \cdot s_e}(Node_{5,t} + D(Node_{2,t} - Node_{5,t} + Node_{6,t} - Node_{5,t}))$$

$$Node_{6,t+1} = \frac{(1-d)}{N \cdot s_e}(Node_{6,t} + D(Node_{5,t} - Node_{6,t} + Node_{7,t} - Node_{6,t}))$$

$$Node_{7,t+1} = \frac{(1-d)}{N \cdot s_e}(Node_{7,t} + D(Node_{6,t} - Node_{7,t} + Node_{8,t} - Node_{7,t}))$$

$$Node_{8,t+1} = \frac{(1-d)}{N \cdot s_e}(Node_{8,t} + D(Node_{7,t} - Node_{8,t} + Node_{4,t} - Node_{8,t}))$$

After the first five time steps, the diffusion field presented in Figure 3.35 is arrived at. Values are rounded after each time step for readability. In the calculations, it can be clearly seen

that all nodes are dependent on the value of the node in position 2 (calculations can be found in Appendix 1).

Fig. 3.35 T=5

First injection

Now that the initial state has been calculated, agents are introduced to the simulation. These agents will begin “hill climbing” towards their highest value neighbour. As they pass through the nodes, they change the diffusion value.

For simplicity, it is assumed that there are five avatars at any given time over each of the nodes in Path A (represented by the black line) for the next five time steps. The results after five time steps are presented in Figure 3.36 (the calculations are available in Appendix 3).

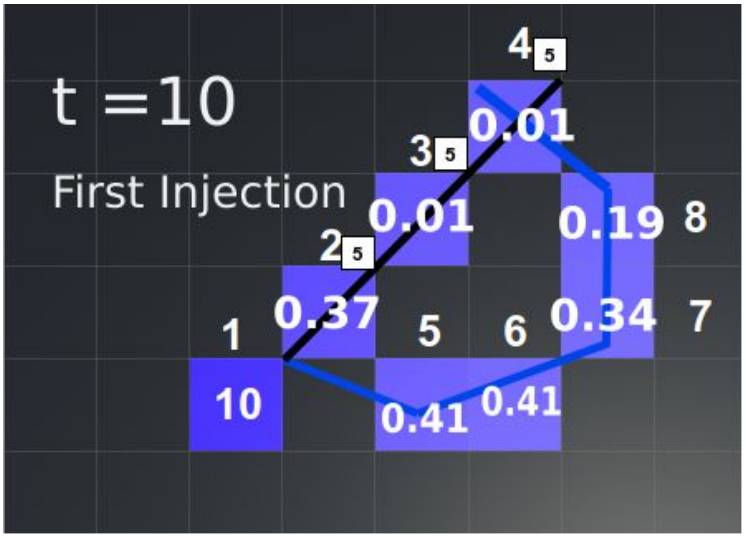


Fig. 3.36 T=10, values in small white squares represent the number of agents on that node

Note that for each node with traffic, the values decrease until reaching the limit of their corresponding diffusion function (Equation 11) within three time steps (Fig. 3.37). In the next

steps, for the sake of simplicity, it will be assumed that the nodes in positions 2, 3 and 4 stay constant.

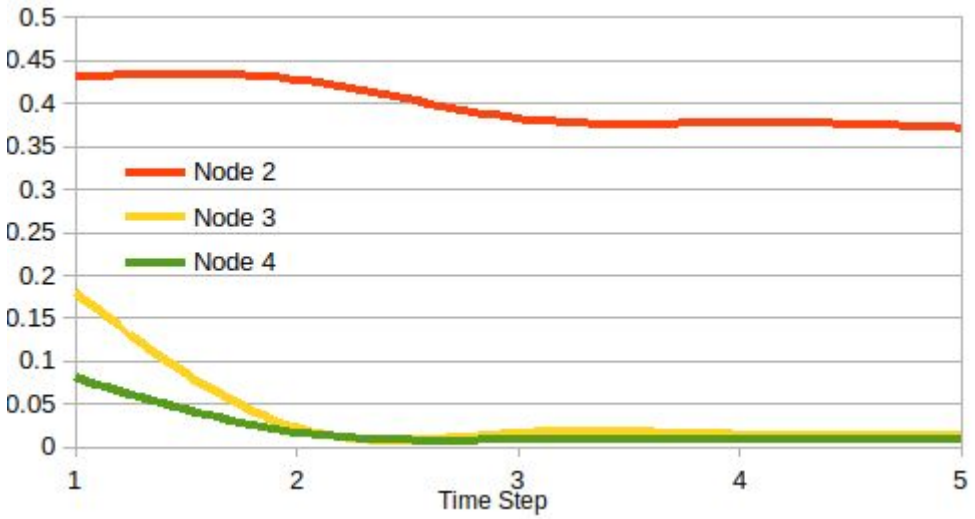


Fig. 3.37 Decreasing diffusion values

Second injection

From this situation, an agent would prefer to start in route B (marked in blue) and go to node 8 instead of node 3. This part will specifically look at nodes 5, 6, 7 and 8 and assume the constant traffic in the other nodes remains. Five agents are spawned at node 4 and they take route B instead of A. This traces what happens when a group of five agents passes through the path, staying for a total of two frames on each node.

This means that at time step 1 and 2 (t=1 and t=2) there are five agents on node 8. Then, they go to node 7 and stay there for time steps 3 and 4, and so on for the next nodes. In reality, this would be more than two frames and would depend on the speed and/or size of the agent and node. (Calculations are available in Appendix 4.) The resulting grid after five steps is presented in Figure 3.38 and the final result after the agents reach their goal after four more time steps is shown in Figure 3.39.

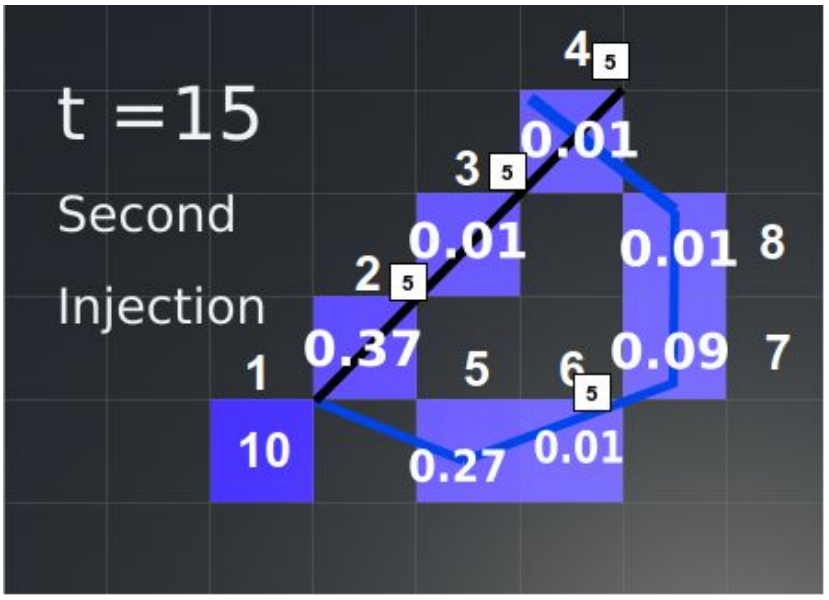


Fig. 3.38 T=15, the value at node 8 is lower than node 3 (written as equal due to rounding)

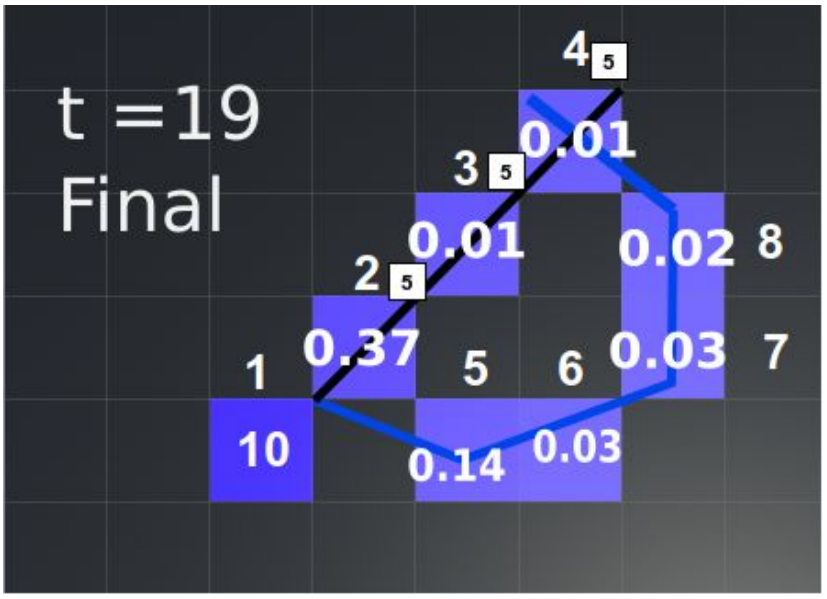


Fig. 3.39 T=19

Conclusion of trace

From this trace it is noticeable that in a closed system such as the one being considered, values further down the chain propagate upwards, which would stop agents from taking this route even if there were no agents on the nodes between them. This is best seen in the second injection at t=15 where the node with traffic blocks the propagation of larger values. Figure 3.38 visualises this. This is in fact a function of the algorithm, but in cases such as these it leads to a suboptimal solution. A method of controlling this propagation is needed.

It is also worth noting that in the covered scenario the fluctuations in node 8 are mainly caused by fluctuations in node 7, 6 and 5 because its only other neighbour (the spawn node at position 4) has an almost constant value. This creates the chain of nodes (path B)

discussed above and shown in Figure 3.38. An attempt was made to formally prove this property, but it was ultimately deemed to be outside of the scope of this paper.

Simulation environment

This part will analyse the same scenario in the Unity testing environment. First, the same field is built, as in Figure 3.40, and the same parameters are set: diffusion coefficient *D*=0.4, decay rate d=0.1, evasion strength $^{S}{}_{e}$=2, with the goal node with a constant value of 10,000,000. Node 1 is the goal and node 4 is the spawn point.

1,000 agents are spawned and begin “hill climbing” the diffusion values. From the trace conclusion the expected behaviour is that the agents sometimes take the uncongested route, but only when there is not a group blocking the diffusion propagation. Thus, the agents should only take path B if there are no groups of agents present anywhere near their last node (which is a proxy for the proxy node).

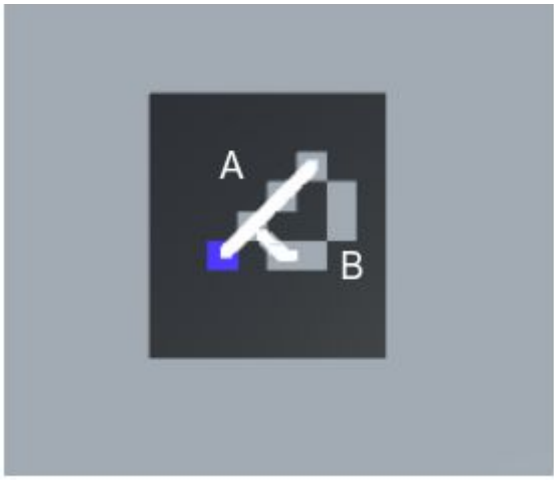


Fig. 3.40 Simulation testing environment test

This is true in the simulation (Fig. 3.40). Agents only take path B when there are no groups of agents travelling on it. The behaviour stays constant despite changes to the diffusion coefficient or evasion strength. For really low values of evasion strength avatars follow path A without affecting the field. The same behaviour was presented in the initial problem for certain segments of the map. Analysing the diffusion values at each node (Appendix 5) shows us how the values continue decreasing from position 5 up to position 8.

Analysis

The bottleneck problem is in fact a well-known problem in chemistry when considering diffusion in a packed bed [27] (i.e. in a directed and cramped environment such as the one in Fig. 3.40). “The ‘bottleneck problem’ refers to the decreases in the rate of solute diffusion due to the presence of ink bottle or narrow channels in the support” [27].

In this case the paths are becoming narrower because of the decreasing of their overall diffusivity by dividing it by the number of avatars currently in that space. This requires a

method of controlling the effects of the propagation, overall decrease of the diffusion values in our grid and the interaction between the diffusion.

Solution

Before solving this problem, some issues with the current implementation and algorithm should be noted. First, as mentioned above, the presented algorithm is intended to work on a tile map rather than a weighted directed graph. The algorithm will not account for the direction of the diffusion pattern (guided by the topology and connections in the graph), nor does it take the weights into account. Moreover, implementing the diffusion equation on a graph is not straightforward, as the partial difference operators need to be applied to weighted graphs rather than just a matrix.

These problems are important because a representation of a road network will always include some kind of direction. Fortunately, diffusion on graphs is quite well understood and many examples exist in lectures, papers and textbooks [32] [36] [25] [22] [30] [28]. This paper does not intend to implement the formally verified and correct solution, but one that would give approximate results and solve the issue being laid out. An attempt will be made to give a general formula that can be implemented in most environments.

To partially address these issues, the current algorithm is slightly adapted from:

$$u_{0,t+1} = \frac{(1-d)}{N \cdot s_e}\left(u_{0,t} + D\sum_{i=1}^{n}\left(u_{i,t} - u_{0,}\right)\right)$$

to:

Equation 12

$$u_{i,t+1} = \frac{(1-d)}{N \cdot s_e}\left(u_{i,t} + D\sum_{u_j \sim u_i} w\left(u_i, u_j\right)\left(u_{j,t} - u_{i,t}\right)\right)$$

Where:
n = number of neighbouring nodes used as input for the diffusion equation
$u_{0,t}$ = diffusion value of the centre node
$u_{i,t}$ = diffusion value of the neighbour node (i>0)
D = diffusion coefficient [0..0.5]
d = rate of decay [0..1]
N = number of agents currently on the node
s_e = evasion strength
$u_j \sim u_i$ = the neighbours (by outgoing edge) of node u_i to all its neighbours
$w\left(u_i, u_j\right)$ = the weight of the edge from node u_i to node u_j

This is done with regards with the Laplacian operator for undirected and symmetric graphs [38]:

Equation 13

$$
\begin{aligned}
(\mathrm{div}_w(\nabla_w f))(x_i) &= \frac{1}{2}\sum_{x_j \sim x_i} \sqrt{w(x_i,x_j)}(\nabla_w f(x_j,x_i) - \nabla_w f(x_i,x_j)) \\
&= \frac{1}{2}\sum_{x_j \sim x_i} \sqrt{w(x_i,x_j)}\left(\sqrt{w(x_i,x_j)}(f(x_j)-f(x_i)) - \sqrt{w(x_j,x_i)}(f(x_i)-f(x_j))\right) \\
&= \frac{1}{2}\sum_{x_j \sim x_i} w(x_i,x_j)(2f(x_j)-2f(x_i)) \\
&= \sum_{x_j \sim x_i} w(x_i,x_j)(f(x_j)-f(x_i)) \;=:\; (\Delta_w f)(x_i).
\end{aligned}
$$

The agent interaction factor is also generalised and added to the continuous version of the diffusion equation. This is included as a general scalar function acting on a matrix:

Equation 14

$Original\ equation$:

$$T'_{i,j} = T_{i,j} + \frac{c_d}{4}\nabla^2 T_{i,j}$$

$Collaborative\ Diffusion$:

$$T'_{i,j} = \lambda\left(T_{i,j} + \frac{c_d}{4}\nabla^2 T_{i,j}\right)$$

Where:

$T_{i,j}$ = a two-dimensional scalar field

c_d = the diffusion coefficient

$\nabla^2 T_{i,j}$ = the Laplacian of a scalar field

λ = agent interaction function that takes in a scalar field, applies the agent interaction and returns the field

Using this adapted formula it is now possible to explore solutions.

Solution 1

The first potential solution researched is to simply adjust the diffusion so that it matches its previous pattern before applying the agent interaction. This will only involve correcting the diffusion so that it conforms to its previous pattern, to include a correction to the pattern that evolves from the agent interaction needed to calculate what the diffusion would have been without the interaction, then find out what the difference is and adjust it. This is presented in the following formula for the continuous equation:

Equation 15

let

$$d\left(T_{i,j}\right) = T_{i,j} + \frac{c_d}{4}\nabla^2 T_{i,j}$$

in

$$T'_{i,j} = \lambda \circ d\left(T_{i,j}\right) + k\left(d\left(T_{i,j}\right) - \lambda \circ d\left(T_{i,j}\right)\right)$$
$$= (1-k)\cdot\lambda \circ d\left(T_{i,j}\right) + k\cdot d\left(T_{i,j}\right)$$

Where:
$T_{i,j}$ = a two-dimensional scalar field
d($T_{i,j}$) = diffusion applied on a scalar field

λ = agent interaction function that takes in a scalar field, applies the agent interaction and returns the field
k = conformity to the original diffusion pattern [0...1]

As seen above, to some extent this is only drawing out the difference of the diffusion values and correcting the applied agent interaction. The implementation of this, however, is not straightforward. The original diffusion value has to always be tracked and compared to what it should have been. This is implemented as:

Equation 16

$$\begin{cases} u`_{i,t+1} = (1-d)\left(u`_{i,t} + D\sum_{u_j \sim u_i} w\left(u_i, u_j\right)\left(u`_{j,t} - u`_{i,t}\right)\right) \\ u_{i,t+1} = \frac{(1-k)(1-d)}{N\cdot s_e}\left(u_{i,t} + D\sum_{u_j \sim u_i} w\left(u_i, u_j\right)\left(u_{j,t} - u_{i,t}\right)\right) + k\left(u`_{i,t} - u_{i,t}\right) \end{cases}$$

Where:
$u_{x,t}$ = diffusion value of the centre node
$u`_{x,t}$ = projected diffusion value of the centre node
k = conformity to the projected diffusion pattern [0...1]

Here, just two values for each node are tracked, *u`* for diffusion that does not interact (projected diffusion) and *u* for diffusion that interacts. The *k* variable allows us to adjust the applied correction. Higher values mean less interaction and lower values mean more interaction. Naturally, the highest value 1 of the conformity factor means that the agents do not interact at all and the lowest value 1 means that the projected diffusion does not affect anything.

This solution works almost perfectly. As seen in Figure 3.41, the agents take both paths and utilise the roads to a higher capacity. The solution also works as expected in open fields; as seen in Figure 3.42, agents take the routes with high values more and ones with lower ones less. This is a simple augmentation to collaborative diffusion and is highly effective in dealing with the described bottleneck problem.

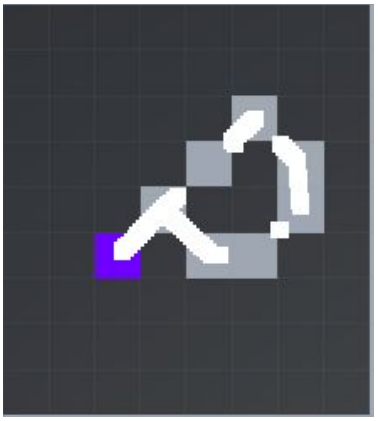


Fig. 3.41 Bottleneck solved by correction

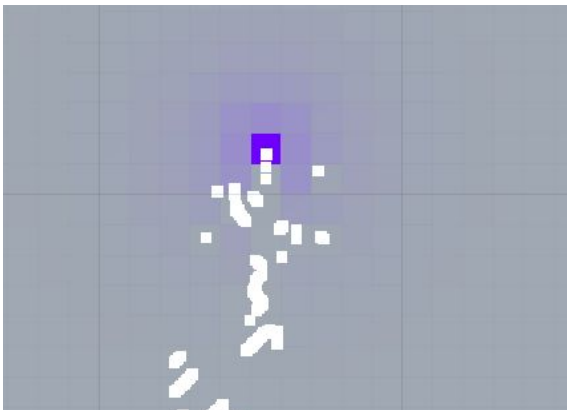


Fig. 3.42 Open field with correction

For most cases, this solution would be entirely acceptable. However, there is a problem with it – the jerk. The concept of jerk is explained in the following quote:

> Velocity does not suddenly switch on, but instead grows from zero. So, there must be some acceleration involved. Similarly, acceleration does not suddenly switch on, but instead grows from zero. So, there must be some jerk involved. But the jerk does not suddenly switch on, it also grows from zero, and so on [39]

In our case, velocity is the diffusion values, the acceleration is the speed the diffusion is spreading and the jerk is the increase in the speed of spreading. This happens because our equation directly adds to the acceleration (i.e. spread of diffusion) of every node, hence jerk is drastically increased. This results in a partially broken diffusion pattern whenever there is a sink (i.e. an agent on a node).

A simple experiment is developed to prove this property. In the testing environment the goal node is placed in the upper left corner, and approximately 10 nodes away 1,000 agents are spawned. As they navigate towards the goal they decrease and increase the diffusion value of the nodes they pass over. In the first run diffusion is used, and in the second our conformity formula with a k (conformity factor) is set to 0.5. Both algorithms are run with a diffusion value of 0.25 and a decay of 0.01.

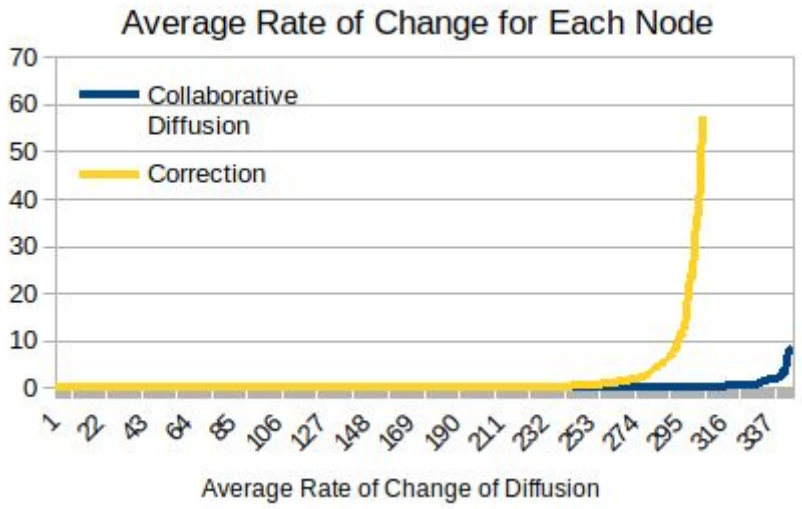


Fig. 3.43. Average rate of change for each node

In Figure 3.43 the effects of the two algorithms on the average rate of change of each node on the grid can be seen. The average rate of change for the conformity-correction group is 1.513 and for the diffusion it is 0.244.

An arising problem from this property is that agents tend to backtrack routes to get to seemingly quicker routes (routes with higher diffusion values) in closed systems. This is because the nodes behind them quickly regain their original diffusion values, which makes them a better path since they are closer or do not have any avatars on them. As a general solution, this would work, but more fine tuning is needed.

Solution 2

To solve the above problem, flow field pathfinding was researched [34] [35]. It is a good solution to simulate the general “flow” of the diffusion values because of its simple implementation and the fact that it again uses the field as a means of finding a path, which is in line with the concept of anti-objects. In the case of flow fields, agents follow the direction to a goal rather than hill climbing (Fig. 3.44).

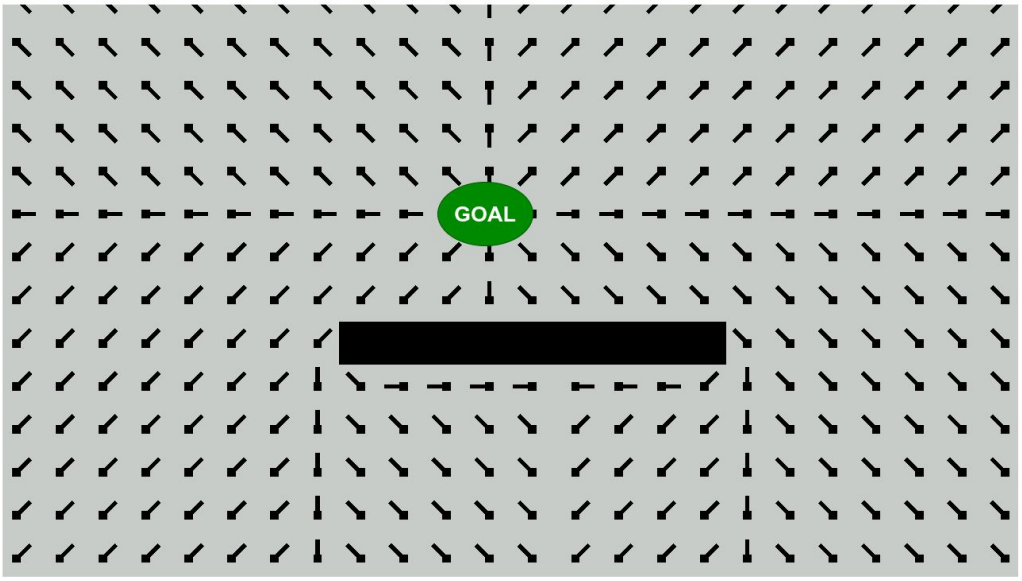


Fig. 3.44 Flow field, the green goal is the goal of the agents. They follow the direction of the vector in their current node [35].

By adjusting the flow, it is also possible to gradually change the diffusion values. Typically, a flow field for pathfinding in a grid (such as the testing environment) is created in the following way:

1. Create the cost field: each node is assigned a value according to its desirability (resistance*)
2. Create the integration field: at each node in that cost field, find the shortest path to the goal for that node (pressure*)
3. Create the flow field: for each of the nodes point to the neighbour with the shortest path (flow*)

After the flow field is created, agents only need to follow the direction at each point to arrive at their goal.

Another way to understand these steps is by considering Ohm's law for flow (Eq. 17) applied at each point of a scalar field. The first step is to create some resistance at each point, i.e. the "flow" at that point is decreased. The second step is to create pressure points so that the air, fluid (or electrons in the case of Ohm) move from higher areas to lower areas of pressure. Step three entails finding the gradient of that whole space, which will give us the direction and magnitude of the flow at each point in space [36].

Equation 17

$$F = \frac{\Delta P}{R}$$

There are three realisations from this:

1. Flow field pathfinding applied directly (i.e. adding diffusion to the flow value) would face the same issues outlined in solution 1.
2. In step two diffusion also moves in the same direction in its concentration gradient: from high to low. The distinguishing feature of diffusion is that it depends on particle random walk (individual particle movement), and results in the mass movement of particles without requiring directed bulk motion. Bulk motion, or bulk flow, is the characteristic of advection [37]. The term “convection-diffusion” is used to describe the combination of the two phenomena.
3. The directed graph does not need to be specifically represented as a scalar field. This idea comes from the fact that the above field is represented as a velocity field rather than just a x,y function.

Knowing this, to solve the problem, implementing convection as a means of spreading the diffusion value (rather than just diffusion) is attempted. First, to get a better understanding of what advection and convection-diffusion represent and how to implement them, this part will look at their equations, specifically advection for incompressible fluids (i.e. assuming a constant density) [32]:

Advection:

Equation 18

$$\frac{\partial \boldsymbol{u}}{\partial \mathrm{t}} = - v \cdot \nabla \boldsymbol{u}$$ [32]

Where:
u = the quantity being advected (a scalar field)
v = a velocity vector field
$\nabla \cdot v$ = the divergence of the vector field v

In the case above, *v* is said to be solenoidal, which would mean that $\nabla \cdot v = 0$.

Convection-diffusion [32]:

Equation 19

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u - v \cdot \nabla u + s$$

Where:
u = a scalar field representing the diffusion concentration gradient (e.g. the distribution of heat)
a = a diffusion constant
$\nabla^2 u$ = the Laplacian of a scalar field
s = a scalar field representing a source or sink
v = a velocity vector field
$\nabla \cdot v$ = the divergence of the vector field v

Note: The convection-diffusion is a combination of the diffusion and convection equations; we can see that the diffusion part $\nabla^2 u$ is acted upon by the advection part $u \nabla \cdot v$.

Implementing this formula would give the desired behaviour. The diffusion term needs to be changed by the lambda function before combining with advection in order to keep the agent interaction. In theory the equation would allow for the smooth control of agent conformity, but there are two major practical issues with implementation, outlined below.

Firstly, there is a condition that $\nabla \cdot v = 0$ for incompressible fluids, which adds additional requirements and checks to the implementation. This is handled in an example in NVIDIA [32] and an implementation is provided for a GPU. Secondly, the algorithm and solutions presented in this paper [32] suggest that implementing convection-diffusion would be too complicated and beyond the scope of adding smooth conformity control. This work is not attempting to simulate the whole flow of a fluid, but rather a gradual influence to the conformity of the agents to follow a path. For these reasons, a simpler solution is sought out.

In a paper researching graphics simulation, Ryo Miyazaki [40] suggests an equation to simulate both flow and diffusion:

Equation 20

$$\vec{v}' = \vec{v} + \frac{k_v}{4}\nabla^2\vec{v} + k_p \mathrm{grad}(\mathrm{div}\vec{v}),$$

Where:
V = the velocity field (represented by a magnitude and a direction)
Kv = the diffusion of the velocity field
$\nabla^2 v$ = the Laplacian of the velocity field
$k_p grad(div \cdot v)$ = the gradient of the divergence of the velocities

Equation 20 describes the viscosity and pressure effect in gasses or liquids. Viscosity represents the resistance or “thickness” of a fluid [41]. It can also be expressed in terms of momentum transport; it is the material property that characterises momentum transport within a fluid, just as the diffusion coefficient characterises heat transport. It is also described by movement from higher to lower regions of concentration.

The equation is separated into two parts, $\vec{v} + \frac{k_v}{4}\nabla^2\vec{v}$ and $k_p grad(div\ \vec{v})$. The first term accounts for the diffusion of the momentum (velocity). It uses the same equation we first introduced for heat diffusion; the Kv factor is the viscosity ration (the same as the diffusion coefficient). The second term accounts for the gradient of the mass flow in the velocity field, i.e. the influence/interaction that the velocity field has on itself. The Kp factor here is the coefficient of the pressure effect that controls how much the pressure affects the velocities.

The equation perfectly captures the preferred behaviour. The graph can be represented as a velocity field and it is possible to use the magnitudes of that resulting field as a replacement for the diffusion values. It is only necessary to add the agent interaction function to the diffusion term, resulting in:

Equation 21

$$\vec{v}^{*} = \lambda\left(\vec{v} + \frac{k_v}{4}\nabla^2\vec{v}\right) + k_p grad(\, div \cdot \vec{v})$$

The theory behind this idea is that the velocity field is diffused and interacted with, but then corrected by its momentum (with regards to its previous direction). It is also useful that the first term (the diffusion one) is practically the same as the one used previously in Equation 9. For the gradient and divergence operator for graphs, discrete calculus gives the following formulas [38] [28] [30]:

Gradient of the graph:

Equation 22

$$\|\nabla_w f(x_i,\cdot)\|_{\ell_p} = \left(\sum_{x_j\sim x_i} w(x_i,x_j)^{p/2}|f(x_j)-f(x_i)|^p\right)^{1/p} \qquad \text{for } 1\le p<\infty,$$

$$\|\nabla_w f(x_i,\cdot)\|_{\ell_\infty} = \max_{x_j\sim x_i}\sqrt{w(x_i,x_j)}|f(x_j)-f(x_i)| \qquad \text{for } p=\infty.$$

Divergence:

Equation 23

$$\|(\nabla_w f)(v_i)\|_p = \left(\sum_{v_j\sim v_i} w(v_i,v_j)^{p/2}\left|f(v_j)-f(v_i)\right|^p\right)^{1/p}.$$

Using these definitions, a draft for an iterative solution of $k_p grad(\, div\ \vec{v})$ is produced:

Equation 24

$$k_p grad\left(\, div \cdot u_{0,t+1}\right) = k_p\left(\, SUM^{n}{}_{i}\left(\, DIVu_{0,t} - DIVu_{n,t}\right)\right)$$

The idea of this solution is that at each iteration the divergence at a specific node in the graph is computed. Then, using these, the gradient of these divergences (the weighted difference of the two values) is calculated. The whole viscosity and pressure equation is built in separate parts. First, the undisturbed diffusion of the grid is computed. At the same time, using the diffusion values of a previous time-step, the divergence at a specific node is computed, while simultaneously calculating the gradient using the divergences at the previous time-step. This calculation is separated into several equations.

First the undisturbed diffusion value is computed. This is then used to calculate the flow of the undisturbed field.

Equation 25

$$diffusion \cdot u_{0,t+1} = (1-d)\left(u_{0,t} + D\sum_{u_j \sim u_0} w(u_0,u_j)\left(u_{j,t} - u_{0,t}\right)\right)$$

Note: The flow $k_p grad(div\ \vec{v})$ is not added to the diffusion here because it is only necessary to project the flow caused by the diffusion onto the interactive field, without changing it.

Then, using the diffusion value and the formula in Equation 23, the divergence at each node is calculated with:

Equation 26

$$div \cdot u_{0,t} = \sum_{j=1}^{n}\left(\sqrt{w(u_0,u_j)}\left(u_{j,t} - u_{0,t}\right)\right)$$

The gradient of the divergence is calculated in the same way as it would be calculated for the nodes [38]. The resulting values for each node from the previous equation are used to compute the gradient, naming this operation flow:

Equation 27

$$flow \cdot u_{0,t+1} = \sum_{j=1}^{n}\left(w(u_0,u_j)\cdot\left(div \cdot u_{j,t} - div \cdot u_{0,t}\right)\right)$$

Then the last equation combines the results of the defined flow operator with the original collaborative diffusion equation:

Equation 28

$$v\&p \cdot u`_{0,t+1} = k_p \cdot flow \cdot u_{0,t} + \frac{(1-d)}{N \cdot s_e} \cdot \left(u`_{0,t} + D\sum_{u_0 \sim u_i} w(u_0,u_j)\left(u`_{j,t} - u`_{0,t}\right)\right)$$

Where:
k = conformity to projected diffusion pattern

Note: The flow term does not need to be scaled by the decay (1-d) as it was computed from an already decayed diffusion. This also uses a new value u`, which is the viscosity and pressure of a node (that replaces the diffusion). The calculations combine both factors and store them in u`, which is used for subsequent computations of V&P. $u`_{0,t}$ could have been written as $v\&p \cdot u_{0,t}$, but for readability purposes it is changed.

To reiterate, the idea of the algorithm presented here is to calculate the general flow of the diffusion pattern and influence agents to conform to that (i.e. change manipulate the agent-diffusion interaction). The resulting Equation 28 should allow for the control of the formation of the diffusion pattern using the additional flow term. It represents the “flow”

caused by the original diffusion field (i.e. before interaction). The conformity to the original field is scaled using the Kp factor. The flow still allows the agents to interact in the same capacity, but it also keeps them going in the general direction of the goal.

The whole algorithm is:

Equation 29

$$diffusion \cdot u_{0,t+1} = (1-d)\left(u_{0,t} + D\sum_{u_j \sim u_0} w(u_0, u_j)\left(u_{j,t} - u_{0,t}\right)\right)$$

$$div \cdot u_{0,t+1} = \sum_{j=1}^{n}\left(\sqrt{w(u_0, u_j)}\left(u_{j,t} - u_{0,t}\right)\right)$$

$$flow \cdot u_{0,t+1} = \sum_{j=1}^{n}\left(w(u_0, u_j)\cdot\left(div \cdot u_{j,t} - div \cdot u_{0,t}\right)\right)$$

$$v\&p \cdot u`_{0,t+1} = k_p \cdot flow \cdot u_{0,t} + \frac{(1-d)}{N \cdot s_e}\cdot\left(u`_{0,t} + D\sum_{u_0 \sim u_i} w(u_0, u_j)\left(u`_{j,t} - u`_{0,t}\right)\right)$$

Note: It is interesting to note that the flow could be calculated for the interactive diffusion value and applied to it; this would cause the agents to form a “flock”.

The implementation of this requires each of the operators to be a separate variable, so it creates variables for: diffusion, div (divergence), flow and V&P (viscosity and pressure). Each variable updates itself without relying on the rest. The idea of implementing it this way is to build a parallel asynchronous implementation for the best results.

In the testing environment the algorithm behaves perfectly and is a definite improvement upon the previous solution. It allows for finer control of conformity with substantially lower turbulence. Agents can be seen taking routes without backtracking in closed environments.

Analysing the average rate of change also suggests the success of the implementation above. To demonstrate the effectiveness, the algorithm is compared to the previous runs, but the flow is set to a high value of 2. This should cause more jerk and increase conformity (relative to the correction algorithm). The other parameters and test scenarios remain the same. The results are displayed in Figure 3.45.

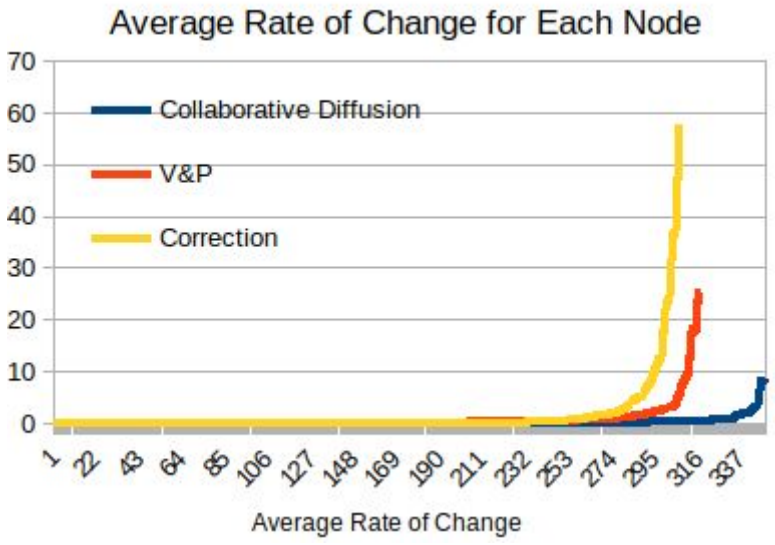


Fig. 3.45

For much higher conformity rates (which would cause more diffusion corrections), there are substantially lower rates of change, suggesting a gradual change in jerk. Table 3.2 shows the average rate of change for each of the algorithms.

Table 3.2 Averages

Collaborative Diffusion	V&P	Correction
0.2450	0.8023	1.5131

Conclusion

Several iterations of solutions were designed to address this problem, but the bottleneck problem was best solved by the two suggested algorithms: collaborative diffusion-correction and V&P. The resulting behaviour fixes the proposed problem to a high degree. The algorithms are also not strictly dependent on the graph data structure; they could be easily implemented on a tile map.

Another notable attempt to solve this issue was to find out in which direction the change was made and to project diffusion in the opposite direction, which would negate changes. This was deemed unnecessarily complicated and outside the scope of the project, however.

The first algorithm is a simple “go-to” solution for the bottleneck problem. It is a simple augmentation to collaborative diffusion that can be implemented with ease. The second algorithm, V&P, is slightly more complex. There are two major issues with the second algorithm as it is written down. The first is that the mathematics involved is not formally validated and is based mostly on interpretation and approximation of how an iterative algorithm solves the issues. This would need to be formally validated and rewritten if wrong.

The second is related and could potentially be drastically simplified into one or two equations.

A potential improvement to the algorithm would be to generalise it so that the flow property is proportional to the error it is trying to correct. This could be a generalised solution for both competition (i.e. when agent interaction is a decreasing function) and cooperation (i.e. when agent interaction is an increasing function). The change here is simple to implement. The addition is that the flow is calculated based on the difference of the two algorithms. For the continuous version, this would be:

Equation 30

$$let$$

$$diff(\vec{v}) = \vec{v} + \frac{k_v}{4}\nabla^2\vec{v}$$

$$in$$

$$\vec{v}` = \lambda(diff(\vec{v})) + k_p \cdot grad \cdot (div \cdot (\lambda(diff(\vec{v})) - diff(\vec{v})))$$

For the discrete version, the only change would be that the divergence is calculated based on the difference of V&P and diffusion:

Equation 31

$$diffusion \cdot u_{0,t+1} = (1-d)\left(u_{0,t} + D\sum_{u_j \sim u_0} w(u_0, u_j)(u_{j,t} - u_{0,t})\right)$$

$$div \cdot u_{0,t+1} = \sum_{j=1}^{n}\left(\sqrt{w(u_0, u_j)}\left((u`_{j,t} - u_{j,t}) - (u`_{0,t} - u_{0,t})\right)\right)$$

$$flow \cdot u_{0,t+1} = \sum_{j=1}^{n}\left(w(u_0, u_j) \cdot (div \cdot u_{j,t} - div \cdot u_{0,t})\right)$$

$$v\&p \cdot u`_{0,t+1} = k_p \cdot flow \cdot u_{0,t} + \frac{(1-d)}{N \cdot s_e} \cdot \left(u`_{0,t} + D\sum_{u_0 \sim u_i} w(u_0, u_j)(u`_{j,t} - u`_{0,t})\right)$$

Equation 31 is the most “correct” form of the proposed algorithm.

A significant benefit of these solutions is that it allows for the parallelising of the whole equation by separately computing the different parts asynchronously (Fig. 3.46 and 3.47), using a CLM approach [22].

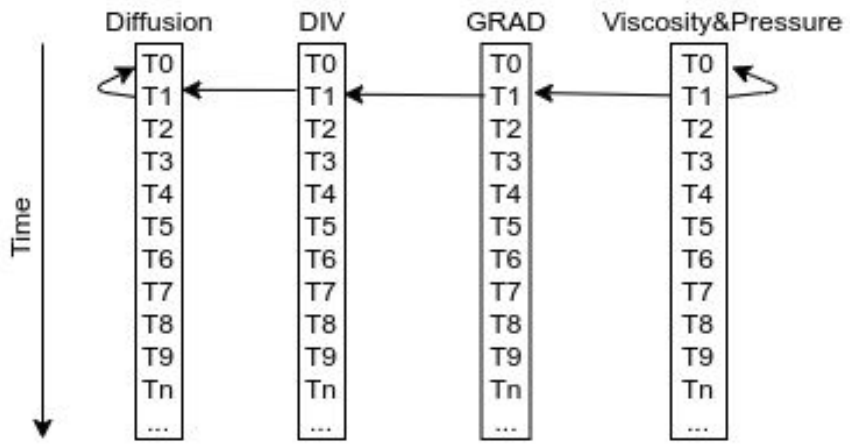


Fig. 3.46. Synchronised concurrent computation. The arrows point to the read values. Each computation here relies on the previous one (i.e. GRAD, t1 relies on DIV, t1 to be computed)

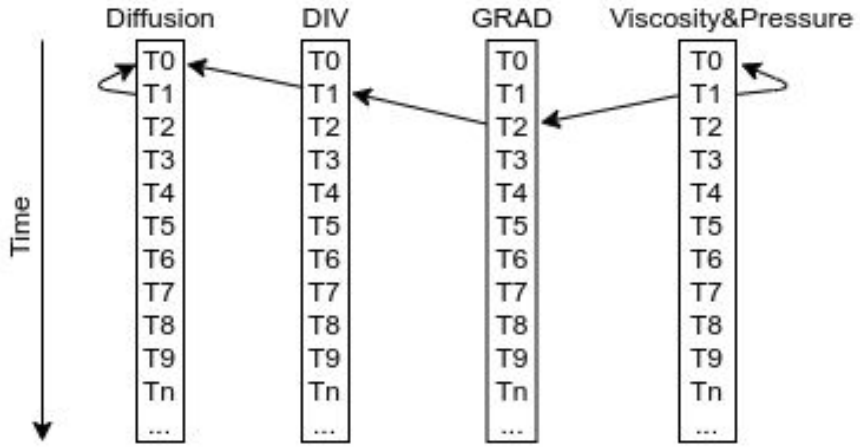


Fig 3.47. Parallel asynchronous computation. The arrows are pointing to the read values. Each computation here does not rely on the previous one (i.e. GRAD, t1 can be computed with previous values of DIV, t1)

Overall, two convincing solutions are produced. Although this section may seem outside the scope of the project, it was deemed essential to attempt to contribute to the area of collaborative pathfinding. The aim of this paper is to raise curiosity and encourage the exploration of collaborative pathfinding and the resulting V&P algorithm.

3.6 Summary

This chapter has introduced the overall design approach used for this system. In summary, the pathfinding framework was significantly enhanced and all milestones were appropriately reached. The resulting class structure for the simulation can be seen in Fig. 3.48. As stated at the start of this chapter, an iterative development approach was adopted. After the compilation of a specific feature, it was thoroughly tested in the testing environment using custom scenarios. Two successful augmentations to the collaborative diffusion algorithm were proposed: collaborative diffusion with correction and V&P.

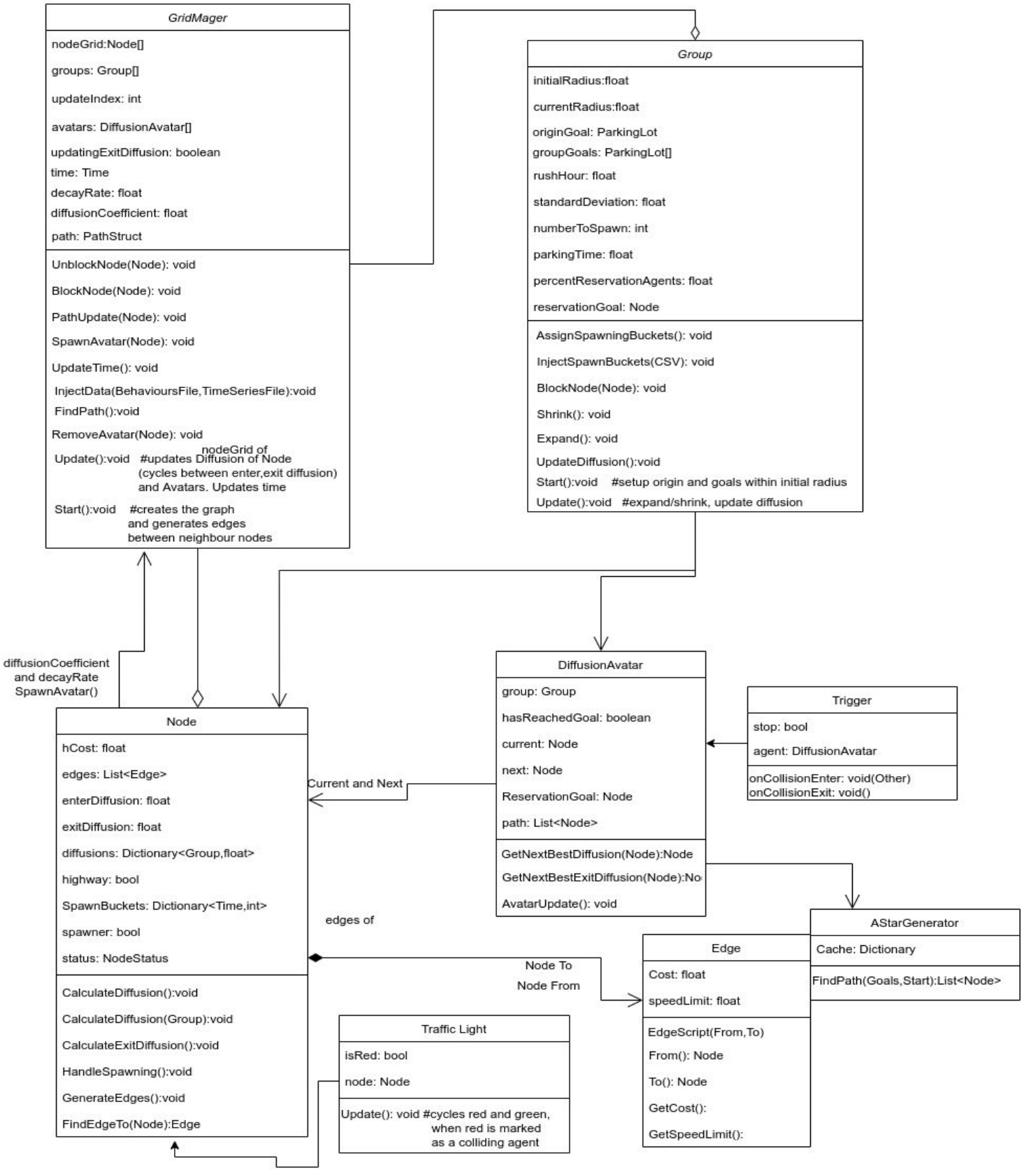


Fig 3.48 Final class diagram

Chapter 4: Results and Evaluation

4.1 Overview

Overall, this project provides a representative mid-fidelity simulation of Newcastle City Centre where the effects of collaborative diffusion on path choice can be seen. The agents take different routes towards their destinations and reduce overall congestion. To test the concept of smart parking using collaborative diffusion as a means of reducing congestion, three experiments are conducted.

The first experiment involves testing the validity of the traffic model using the defined traffic metrics. As the number of agents increases, congestion markers are expected to increase as well. The second and third experiments provide the proof of concept for the collaborative diffusion algorithm. A comparison is made between the collaborative algorithm and non-collaborative shortest path solutions, namely Pythagoras A* and non-collaborative diffusion.

These comparisons are made in two scenarios, one with low traffic and one with high traffic. All traffic data used in these scenarios is based on vehicle count sensor data obtained from the Urban Observatory [11]. Before any informed conclusions can be made regarding results, the methods and metrics should be explicitly defined.

4.2 Definitions and Methods

A combination of standard methods for measuring urban congestion are employed, such as those used in the Freeway Performance Measurement System (PeMS) [42], Urban Observatory [11] and SCOOT variables [43].

Vehicle count: vehicle count is defined as the number of agents in the simulation at any given time. It is interpreted as equivalent to the vehicle count measurement in the Urban Observatory data and is mainly used to inject the extracted vehicle count into the simulation.

Flow: the number of frames of an agent over a node is defined as flow (agent per frame). Each node contains its own flow value over time. Flow is generally detected in real life traffic scenarios using inductive loop detectors or, in the case of urban observatory number, plate recognition cameras. It is widely used as an overall traffic indicator [42]. This metric is used to determine the traffic in the simulation by aggregating the flow from every node in the system. A high flow value indicates a high level of congestion.

Agent travel time: in the simulation, agent travel time represents the summation of the time taken for each agent to arrive at its respective goal. This is used to compare the relative increase or decrease in the additional time taken by the pathing algorithms in urban congestion environments. It is analogous to the standard traffic metric of VHT (vehicle-hours travelled) [42] [43].

Agent path length: this is the summation of each agent's route length. In a real-life traffic scenario this would represent VMT (vehicle-miles travelled). Agent path length is used to

compare the path efficiency of the two algorithms. It is also used in combination with agent travel time as an indicator of the extra distance vehicles have to travel as a consequence of avoiding congestion [42].

Road occupancy: this is the sum of the occupancy/capacity for each road at each hour. It is measured by first creating a road capacity value for each edge between two nodes based on the Pythagoras distance of the nodes. Then at each point that any agent enters or exits the edge, occupancy is added or subtracted from the edge and the percentage of occupation is calculated by dividing occupancy by capacity [43].

Peak congestion point: the peak congestion point is equivalent to the node with the maximum flow value throughout the entirety of the simulation. It highlights the location most vulnerable to congestion. The higher this value, the more likely a given route increases congestion throughout the whole road network. This is analogous to choke points or traffic jams.

4.3 Experiments

A. Simulation validation

First, the validity of the model is tested by testing how the simulation reacts to different traffic levels. In particular, the following markers are expected to be affected by increases in traffic:

- Flow – as a general indicator of traffic, it should be directly dependent on the number of agents.
- Peak congestion point – as flow increases, the bottleneck of traffic should become better defined.
- Road occupancy – an increase in traffic is expected to lead to more agents travelling on the same road at the same time.
- Agent travel time – an increase in road occupancy and flow should in turn decrease the speed of agents due to constant “stop-and-go” behaviour when they are travelling on congested roads [43].

Two different sets of data obtained from the Urban Observatory are used for the scenarios of low and high traffic. The average daily vehicle count in Newcastle City Centre from the 24th March to the 22nd April (during the COVID-19 lockdown) is used to simulate low traffic and high traffic, the vehicle counts for the whole year of 2019. Both sets are the summation of the average daily vehicle counts for each sensor in the city centre. In Figure 4.1, the difference in daily vehicle count can be clearly seen. More specifically, there is a 70% overall change from the annual average [44] for the low traffic group.

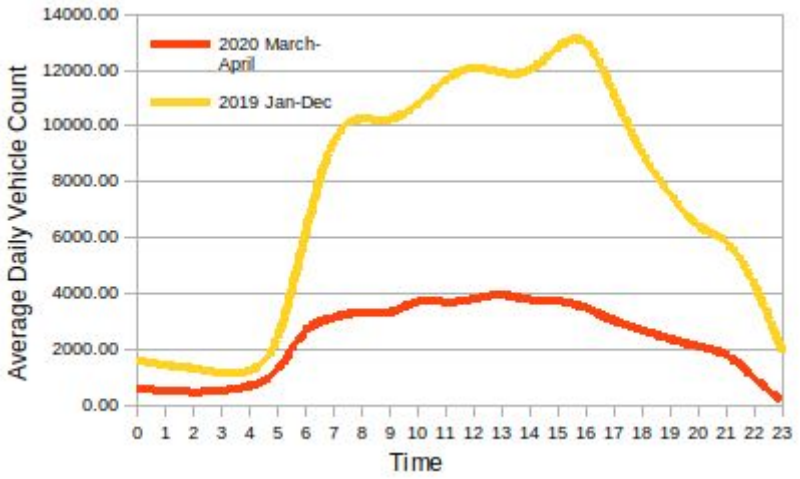
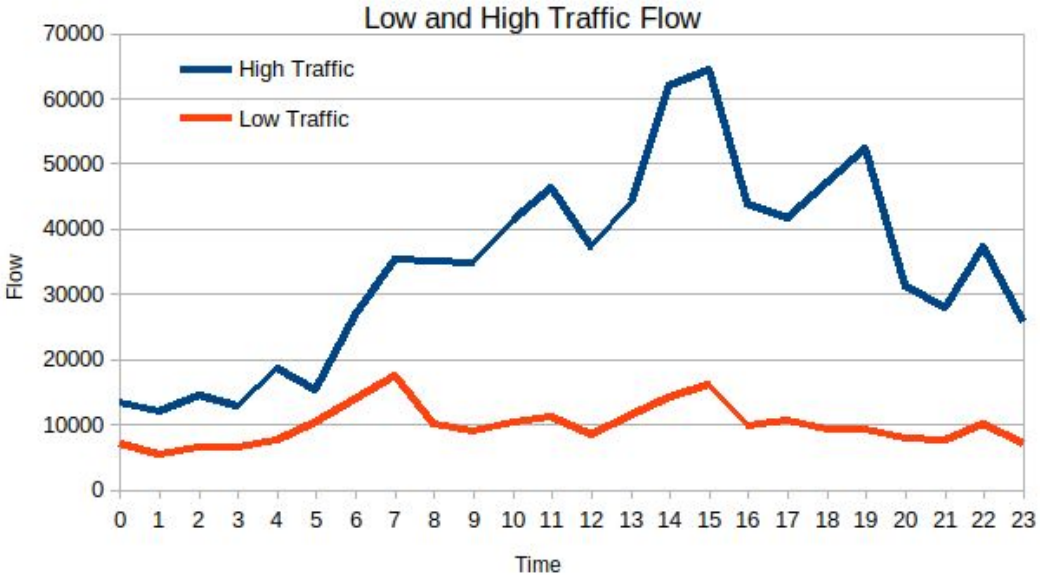


Fig. 4.1 Average daily vehicle count sums for city centre sensors

This data is injected into the simulation with simplified common behaviours of commuters modelled [45] [46] (behaviours and scripts can be found in Appendix 6). Both scenarios are made using A-Star pathfinding, with the goal of exaggerating the differences in the specified metrics. The “inject data” option is used to set the spawn times and behaviours of the different groups. To decrease random factors that might affect the results, multiple runs are done and the average is taken.



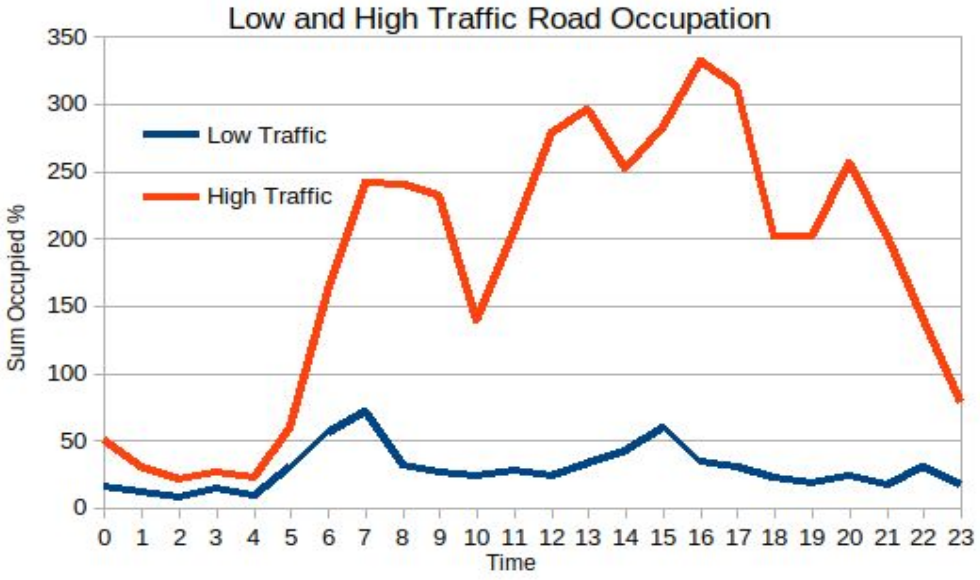


Fig. 4.2

This part will begin by analysing the main metrics of traffic: flow and road occupancy. In Figure 4.2 it can be seen how flow changes over time in the simulation. The y-axis shows the flow value and the x-axis the time of day. In Figure 4.2 it is also possible to see the road occupancies for both groups, where the y-axis shows the road occupancy for that specific hour. For both flow and road occupancy the data from all entries for specific times was computed and added to 24 buckets (for the hours of the day). From the charts, it is clear how the injection data roughly relates to the flow and road occupancy values.

With regard to flow, the low traffic injection generated 71.175% less flow than the high traffic injection, which aligns with the actual data. This was expected simply because of the number of vehicles. It is worth noting that vehicles increase the flow much more quickly when they are standing on a node and not moving, compared to just passing it. It is expected that flow

will be lower for the groups with less congestion. It is possible to conclude using this metric that the simulation accounts for congestion relatively reliably.

For road occupancy, the low traffic group generated 84% less occupancy than the high traffic group. This indicates that the number of vehicles travelling on the same roads at the same time increased substantially when the number of vehicles increased.

As seen in the charts, the distribution of the flow data slightly differs from the distribution of vehicle counts in the urban observatory data. To get a better appreciation of why this is, the distribution of vehicle count in the simulation compared to flow rates is explored.

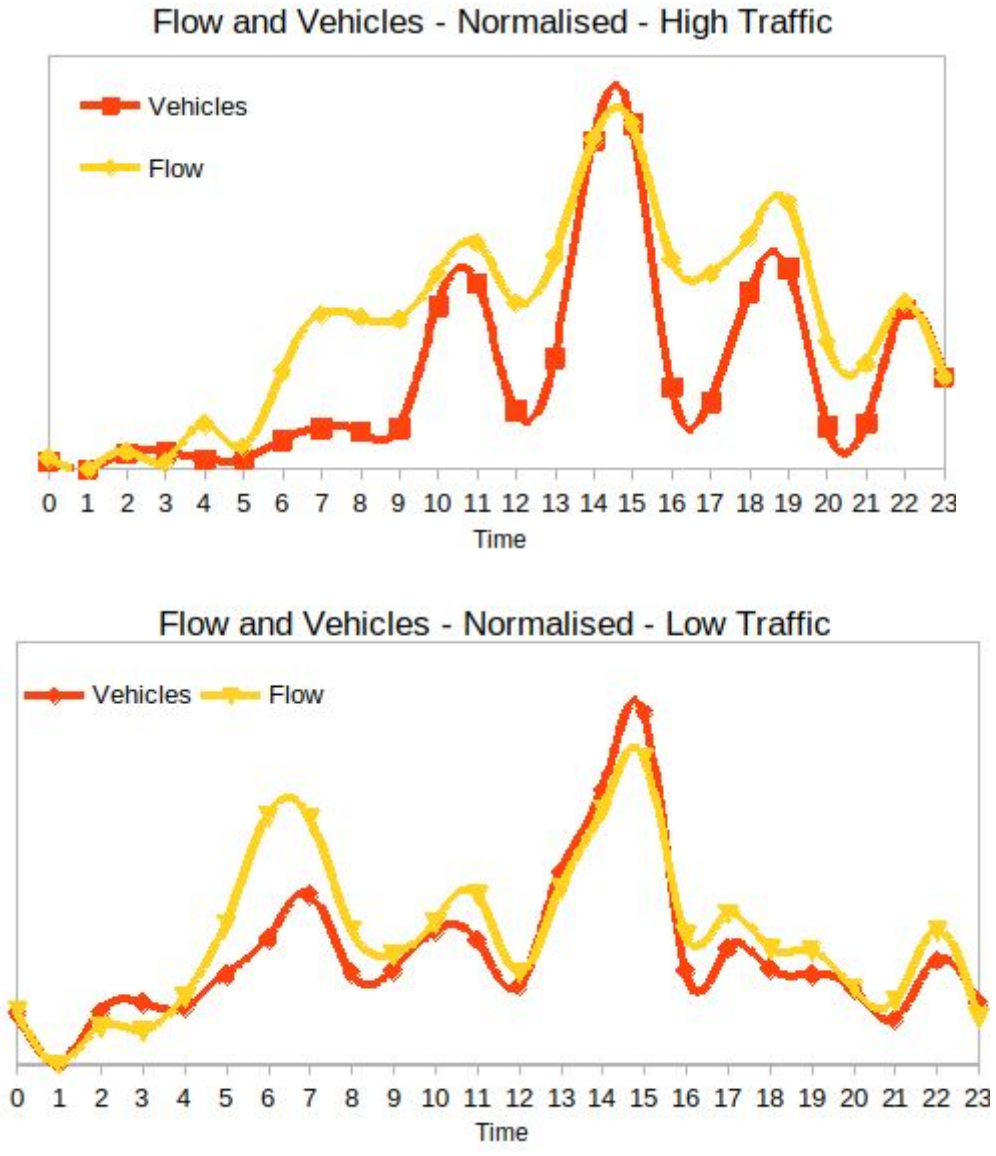


Fig. 4.3

Figure 4.3 shows the distribution of flow and vehicles for the low and the high experiments. The x-axis shows the hour of the day and the line-chart shows the distribution of flow and vehicles throughout the day in the simulation. The vehicle count distribution is largely different from the one displayed in the original data in Figure 4.1 because of the simple behaviours the data is injected with.

As can be seen in the graphs, flow rates are directly affected by the number of vehicles. There are only slight problems with some places, such as 06:00 in the low traffic where it is possible to see disproportionate increases in flow. This is attributed to a potential error somewhere in the code responsible for counting flow. Overall, however, flow is affected by all behaviours on the road and this means that it is a better approximation of car movement than vehicle count.

Another metric that is directly related to flow is the peak congestion point – the highest flow value. This is presented in the heatmap below. This shows the distribution of flow in the simulation. The point of highest flow for the high traffic group is twice as large as the one for the low traffic group. This is the expected behaviour and it validates the point of highest flow metric.

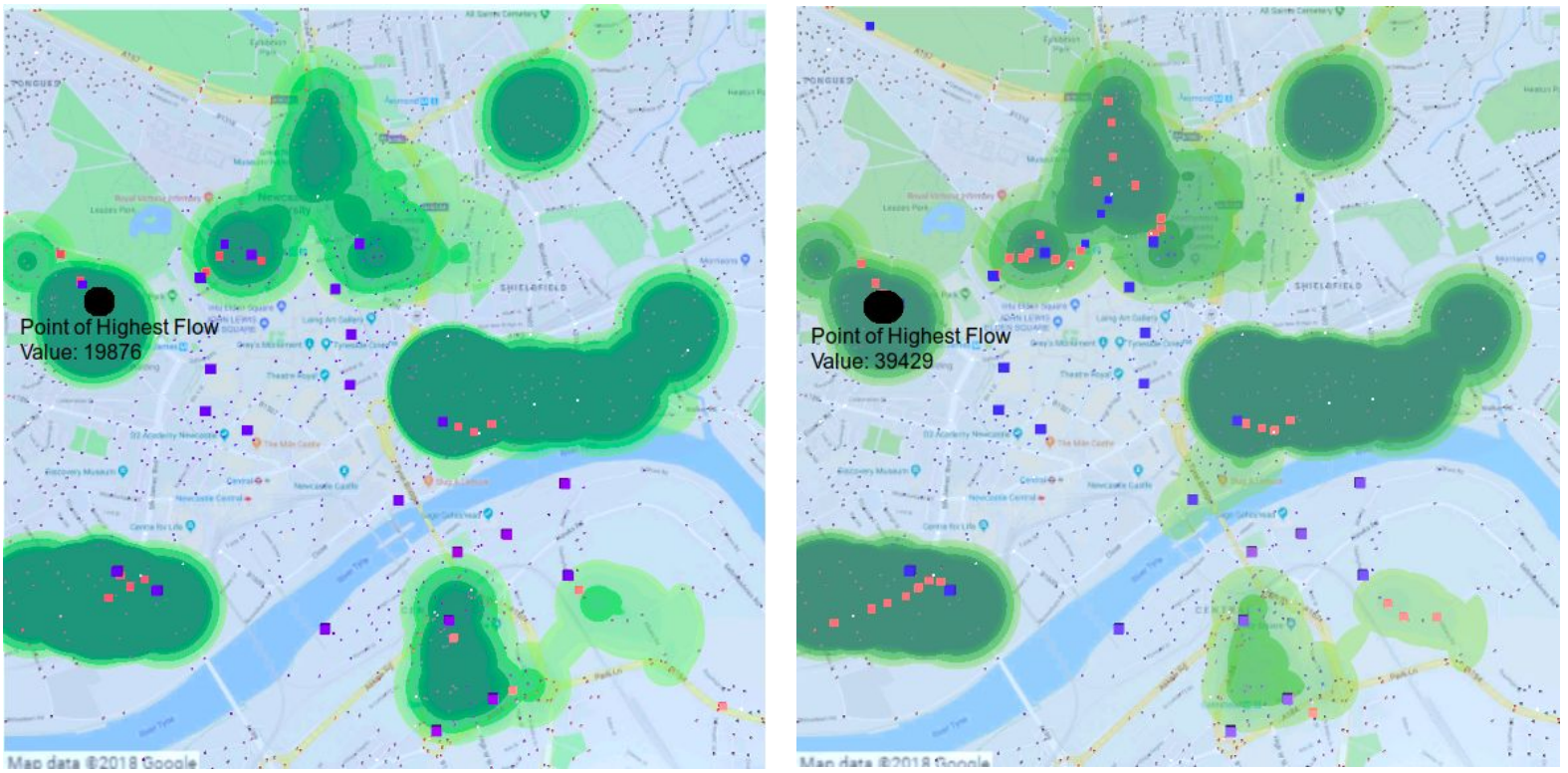


Fig. 4.4 Heatmap of flow – low traffic on the left, high traffic on the right

On both maps, it can be seen that the distribution of the flow values is relatively the same (i.e. the heatmap patterns are similar). This is because A* pathfinding is being used for both runs, and the vehicles are using the same routes to their parking lots. Different patterns are expected to develop for the different algorithms.

In the two maps it is possible to see that the bulk of the traffic is near the edges and the closest goals to the spawn points. This is due to the simplistic behaviours that the tests are run with and the fact that A* directly finds the shortest path towards a goal. From this, it can be determined that the peak point is affected by traffic.

The most telling metric employed is VHT (or agent travel time), since it should be directly affected by congestion. The chart in Figure 4.5 shows the average agent travel time for each of the 24 hour time slots. The x-axis is time and the y-axis is the journey time in seconds.

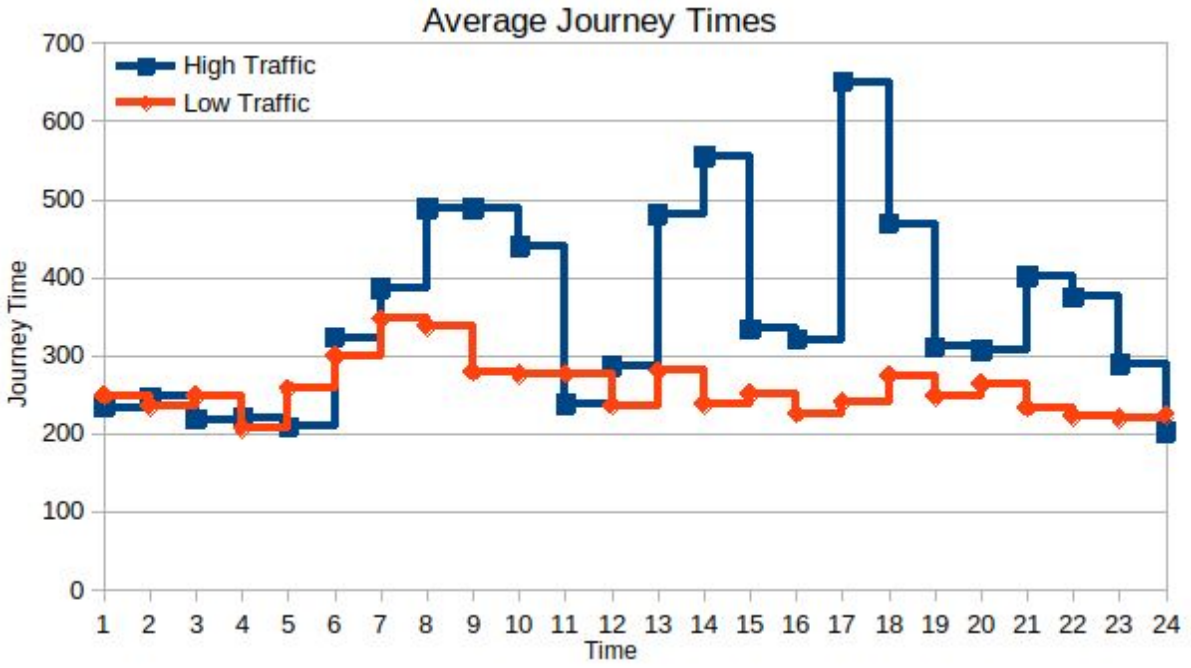


Fig. 4.5

The distribution of journey times is similar to those seen in flow and road occupancy. The average journey time for the low group is 257 seconds, while in the high group it is 353 seconds. The low traffic group has on average 27% lower journey times than the high traffic group. It is thus clear that journey time increases as traffic increases. The effect that traffic has on journey time can be tuned further by changing the size of the vehicle/collision boxes or by changing how the collision system works. However, how does the generated data compare to data from Urban Observatory?

Although the main aims of the project do not include simulating accurate traffic results for real-life scenarios, an interesting parallel is drawn between the simulation and real traffic. For the collaborative diffusion solution, the high traffic scenario generates a similar flow to the one seen in the Urban Observatory data. This is best visualised in a heatmap comparison (Fig. 4.6).

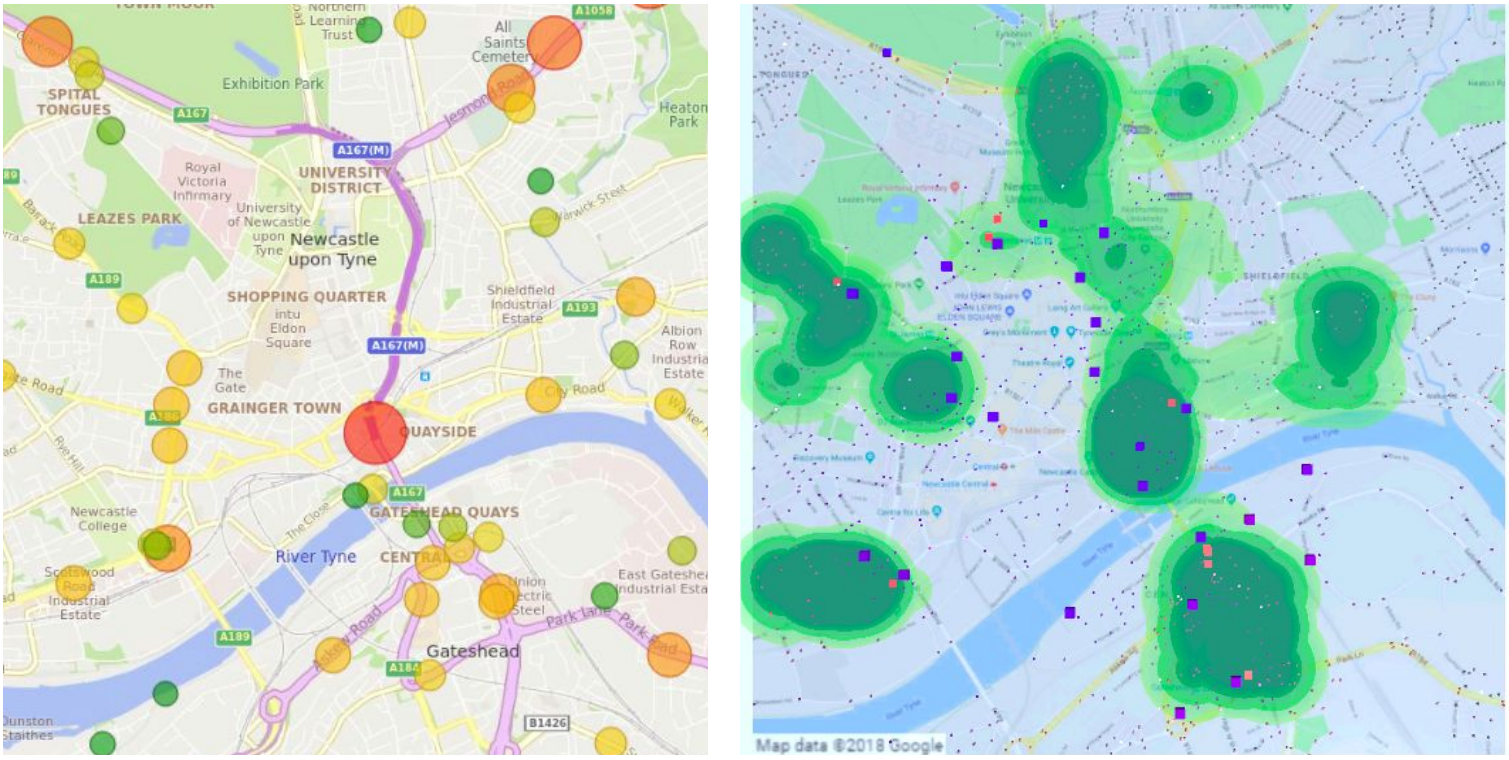


Fig. 4.6 Heatmap comparison – on the left is data from the Urban Observatory, whilst the heatmap on the right is the results from non-collaborative diffusion

The distribution of traffic clusters is relatively similar in the two datasets. This is presumed to be due to the topology of the road network and the placement of traffic lights. As seen in the two scenarios, the positions of the majority of the bigger clusters are in locations with traffic lights.

However, looking at other data suggests that the simulation is inaccurate in some regards. This seems to be especially the case for the low traffic datasets. Figure 4.7 shows a comparison between the agent travel time (journey times) of the simulation and the real-life data. Note that the actual values of agent travel time from this experiment were divided by two and then presented (e.g. actual value for the simulation journey time at 04:00 is 200 but it is presented as 100). This difference is assumed to be mainly due to inaccurate traffic light timings. The mean of the adjusted simulation journey time is 128 seconds, while the mean for the Urban Observatory data is 118 seconds.

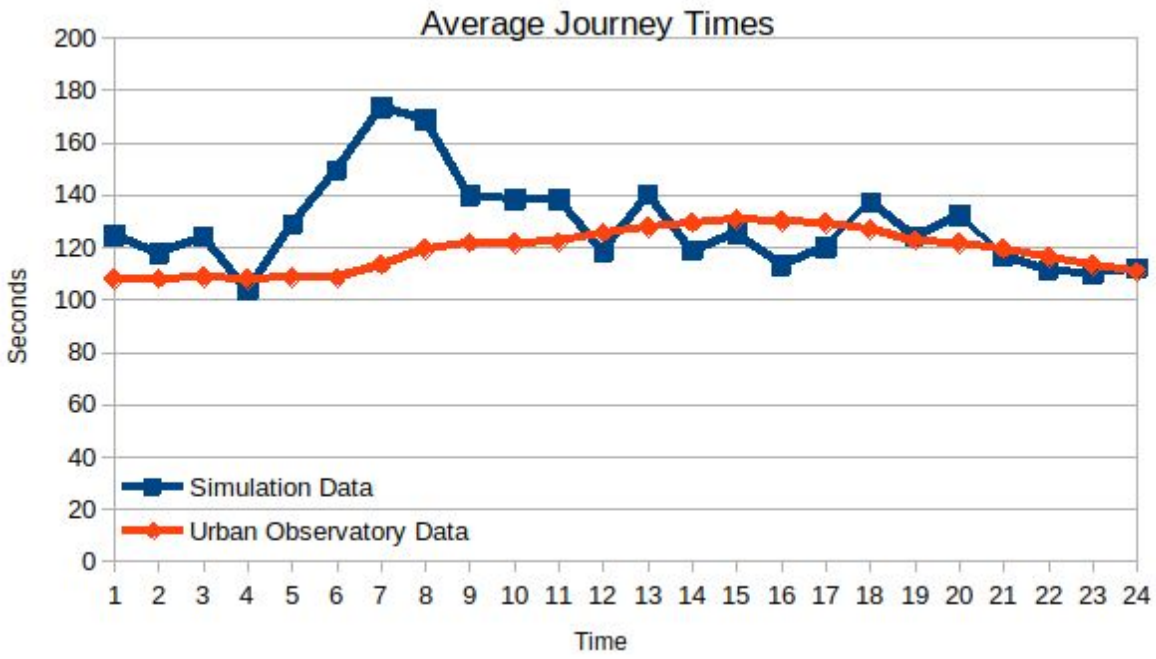
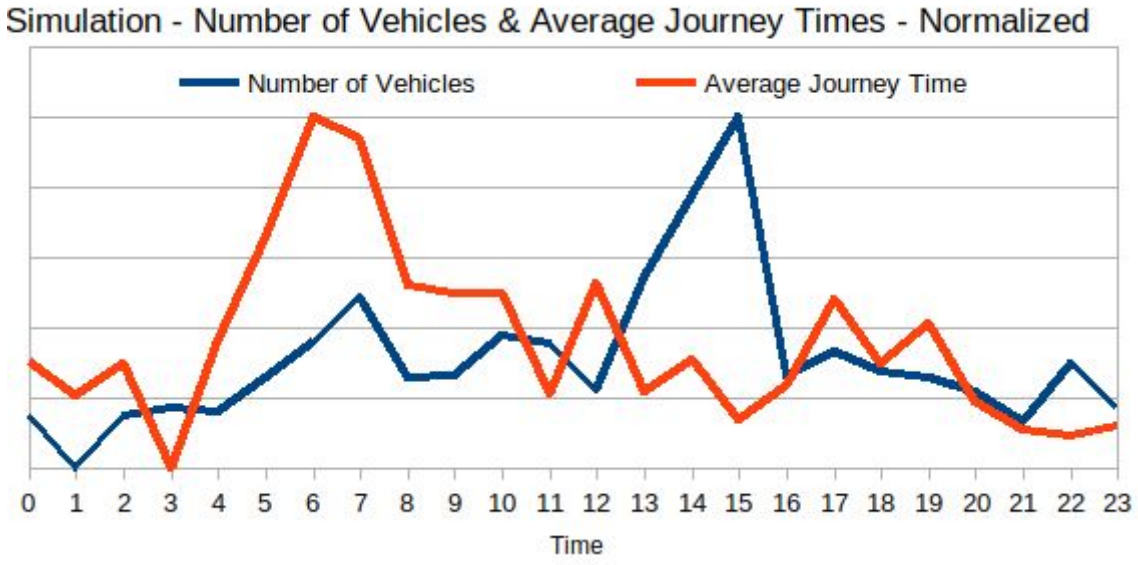


Fig. 4.7

The distribution of the journey times throughout the day is also inaccurate. There is an increase at 07:00 where the simulation data reaches a peak. As described above, this is attributed to inaccurate behaviours rather than wrong traffic modelling, as we can see the same spike in flow at 07:00 in Figure 4.3. To explore whether this is the case, vehicle counts and agent travel times are both normalised and their relative distributions are analysed for both the real-life data and simulation, as in Figure 4.8.



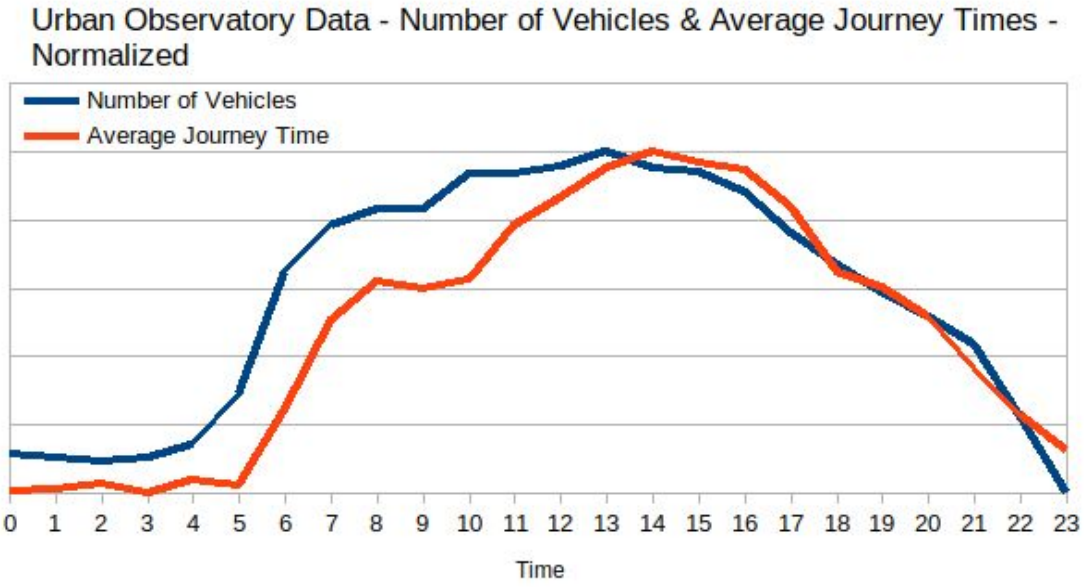
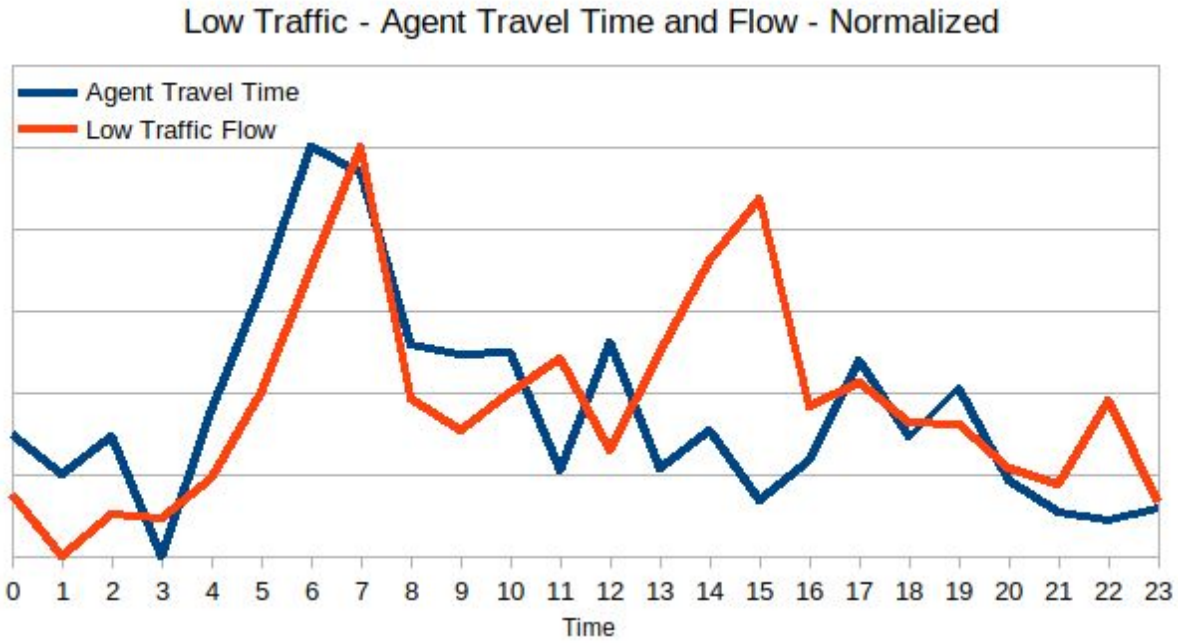


Fig. 4.8

The real-life data shows how increases in vehicle counts relate to increases in the average journey times. For the dataset this is not the case: the two variables differ greatly. This suggests that there is an error in the traffic modelling rather than the specified behaviours. There are also obvious differences in the movement of vehicles (i.e. flow) and the average journey times as well (Fig. 4.9), which further suggests an error. This problem also occurs in the high traffic injection (Fig. 4.9). However, there is a general trend that journey times increase as vehicles increase in the majority of the cases, hence the mean journey times and the metric itself are still reliable.



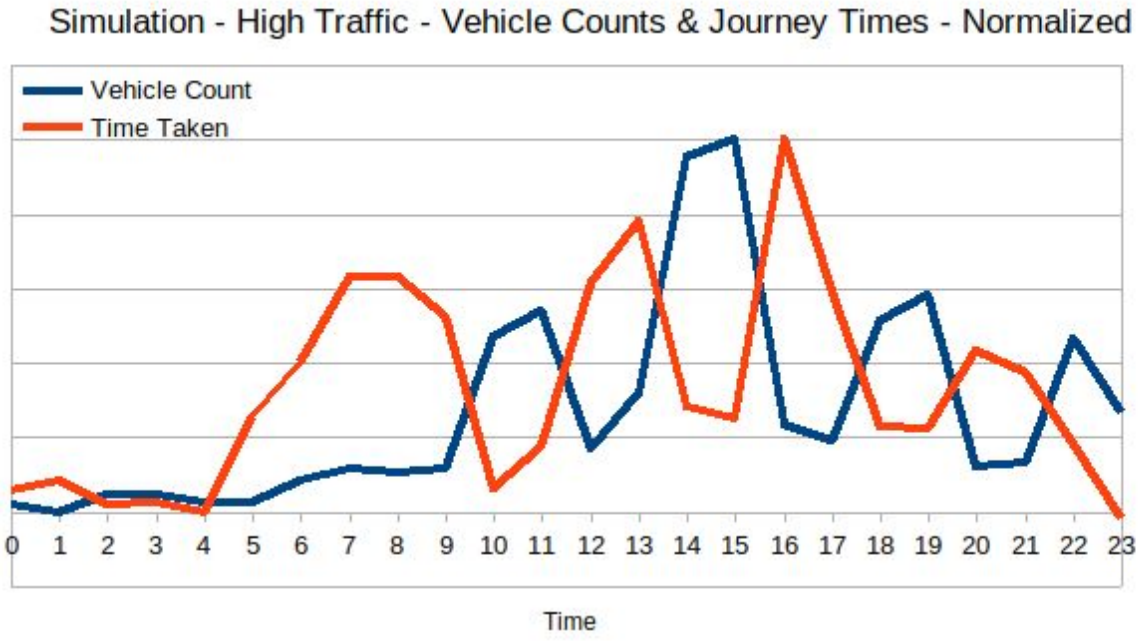


Fig. 4.9

B. Low traffic experiment

In the low traffic experiment we inject the March-April data (Figure 2 from the previous experiment) and compare collaborative and non-collaborative pathfinding solutions: A* Pythagoras, non-collaborative diffusion (NCD) and collaborative diffusion (CD). The traffic level starts increasing at 06:00 and remains relatively constant until 18:00, after which time it decreases. Due to the low volume of traffic, this scenario presents the best-case environment for the non-collaborative solutions.

We note that the A* algorithms are compared directly to CD, using the described measurements. The NCD solution is compared to CD to determine whether the collaboration between agents is responsible for the congestion results rather than the topology of the grid and diffusion. Due to the low traffic rates (i.e. no collaboration needed), this is the best case scenario for A* pathfinding.

In Figure 4.10, it is seen that there are no major differences between the algorithms in terms of flow and road occupancy. In the charts, the average flow or road occupancy values for all algorithms over the number of vehicles in the simulation at the same time is shown. To illustrate, the number of vehicles at 05:00 is matched with the flow value at 05:00. The results are then ordered. The idea is to show how the algorithms react to increases in number of vehicles.

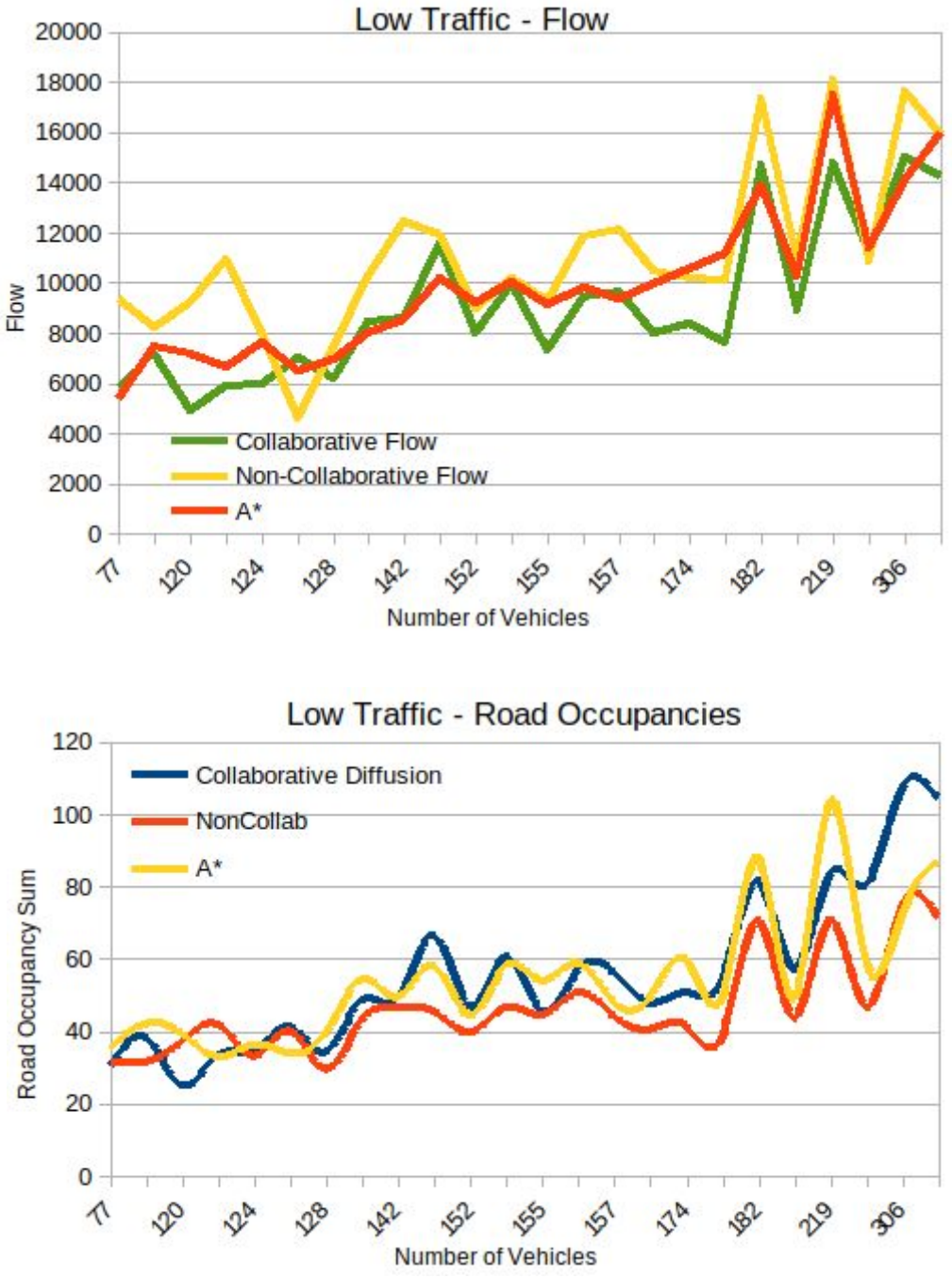


Fig. 4.10

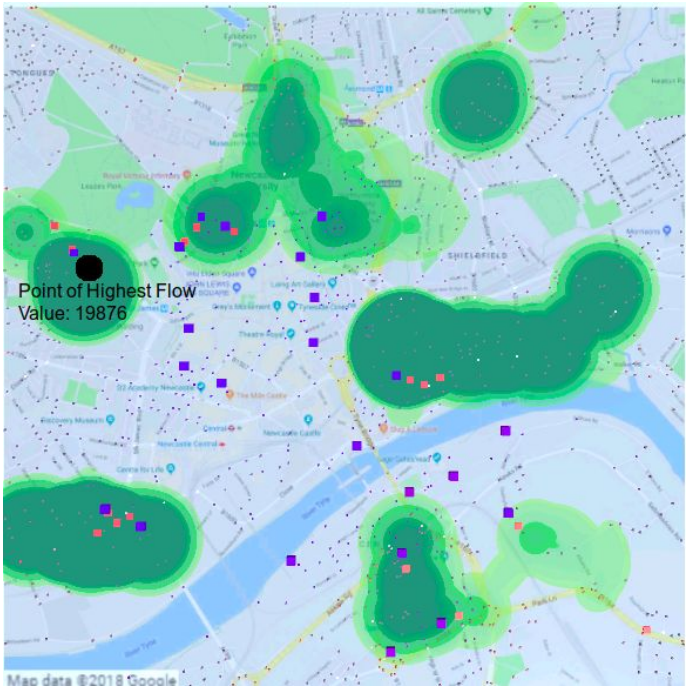
For average road occupancy, there is only a slight difference between A* and CD. With regard to NCD, this is significantly less than the other two solutions. This indicates that agent interaction in collaborative diffusion could potentially be creating more congestion, rather than alleviating it.

The average flow of collaborative diffusion is slightly lower than the flow of A*, while the non-collaborative diffusion solution gives the worst result. This is presumably the case because the NCD solution gives a worse path than A* and it does not take into account other agents as CD.

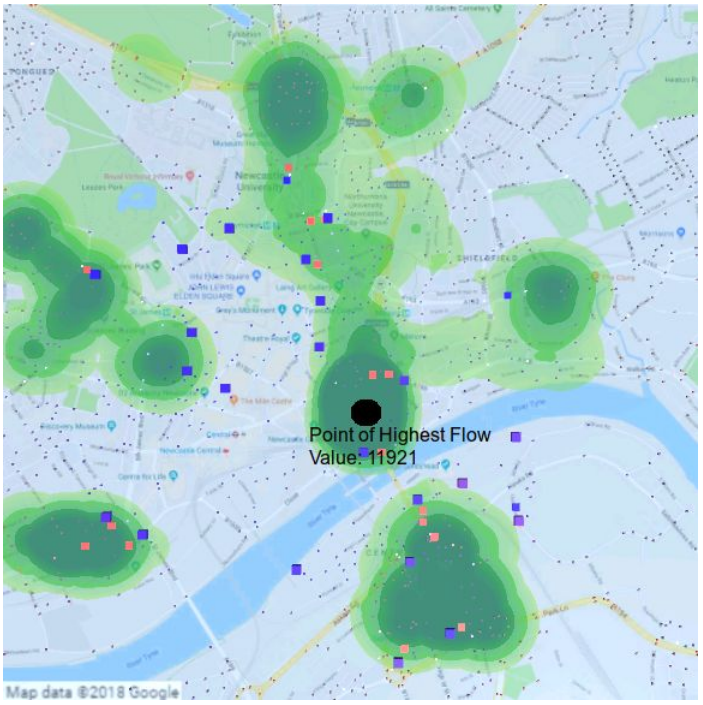
Table 4.1

Averages of	A*	Non-collaborative Diffusion (NCD)	Collaborative Diffusion (CD)
Road Occupancy:	54.3	31.6	55.6
Flow:	9,860	11,101	9,125

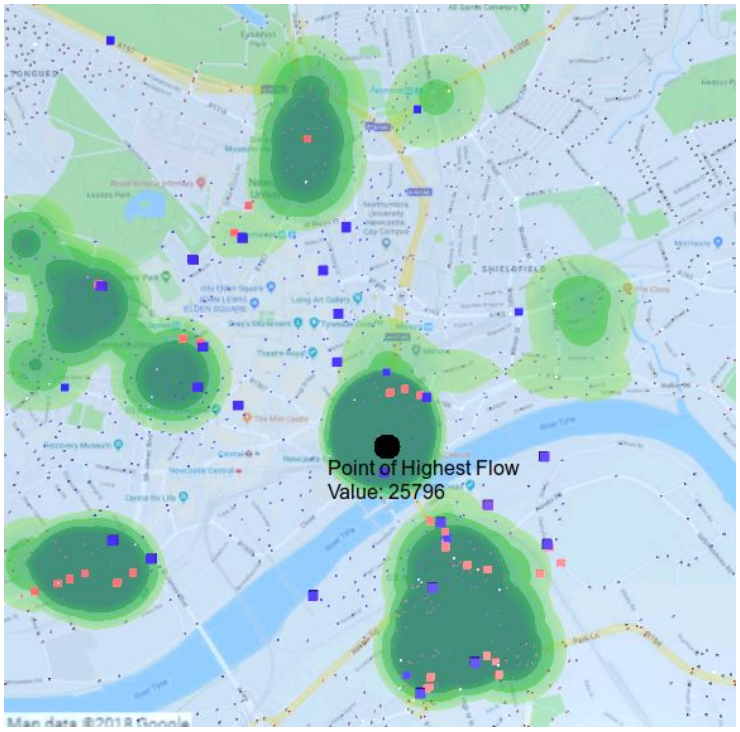
This is also the case when looking at the heatmaps of the algorithms (Fig. 4.11). CD is seen to “spread” out the traffic, while more concentrated clusters can be seen for both A* and NCD. Interestingly, the peak congestion point for CD is 40% lower than that of A*. This is attributed to the success of agent interaction, since we can see that without it in the NCD case we have much higher traffic than with the other two algorithms.



A*



Collaborative Diffusion



NCD

Fig. 4.11

When looking at the pathing cost and travel times, a significant difference is seen between the algorithms. As expected, this scenario proves to be the best case for A*. The travel times do not correlate with the number of vehicles or time, as previously discussed, and this is further proven when analysing how travel times change for the different algorithms (Appendix 7). The average values, however, show that A* generates shorter paths and allows agents to arrive much more quickly at their destinations. For CD, agents arrive slightly faster than they do for the NCD solution, but there are no significant differences between the two algorithms (as seen in Table 4.2).

Table 4.2

Averages of	A*	Non-collaborative Diffusion (NCD)	Collaborative Diffusion (CD)
Travel Time (seconds):	257.26	305.13	291.58
Path Cost:	118	171	169

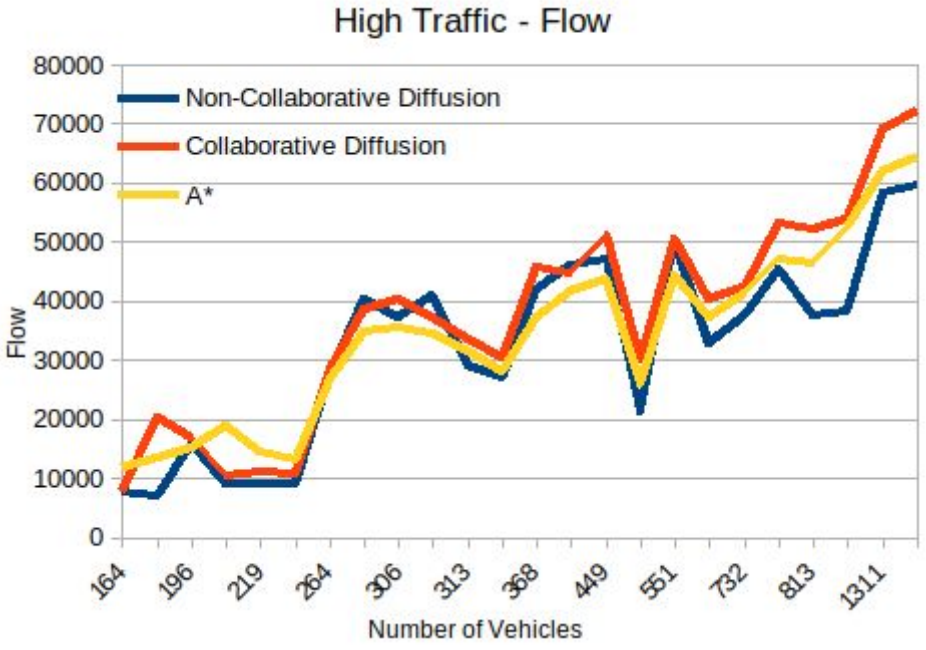
The similar values for path cost (Appendix 8) indicate that the agent interaction was not a major influence in this experiment. This is concluded because NCD shows CD without interaction.

C. High traffic experiment

For the high traffic scenario, similar to the low traffic experiment, traffic data is injected for the 2019 period. Here the traffic increases at 06:00 and continues to increase until 18:00. The aim is to present a high traffic scenario where many agents simultaneously navigate towards their destinations. This is expected to be the worst-case environment for the non-collaborative algorithms and the best-case for CD. The same algorithms and methods are employed as in the previous experiment. In this experiment significant differences are expected between NCD and CD due to the number of agents.

The generated flow and road occupancy in this run have stark differences. First, CD has the largest amount of flow of all algorithms, followed by A* (Figure 4.12). The lowest flow is found to be from NCD, which also has substantially lower road occupancy than both algorithms. This is accredited to the undisturbed diffusion pattern formed by the topology of the grid, and the fact that CD causes agents to go around the city until an uncongested path is available. This is further supported by CD having the largest road occupancy of all three groups.

Averages of	A*	Non-collaborative Diffusion (NCD)	Collaborative Diffusion (CD)
Road Occupancy:	207.2	157.7	238.2
Flow:	34,208	32,300	37,098



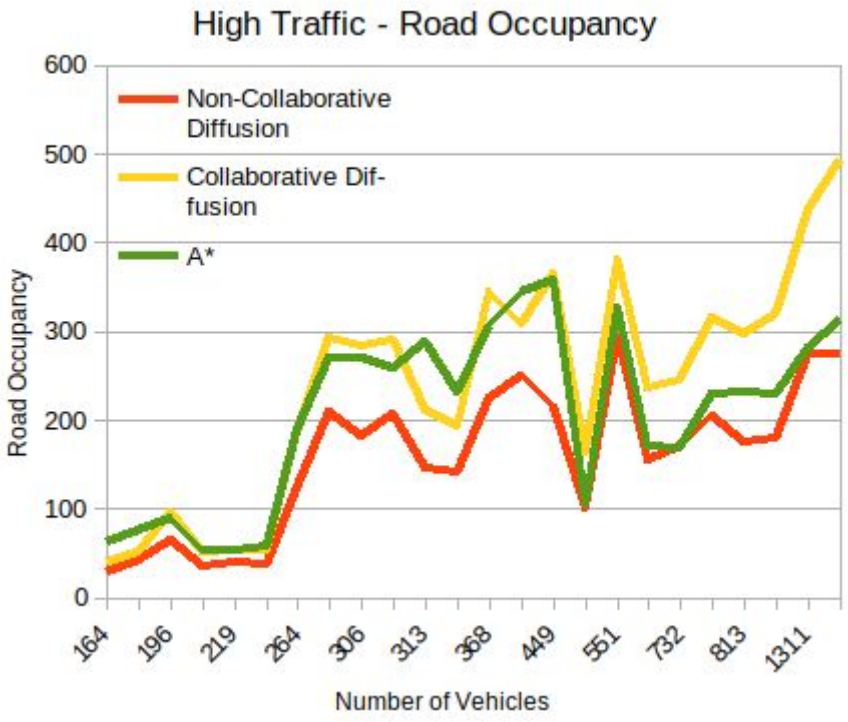
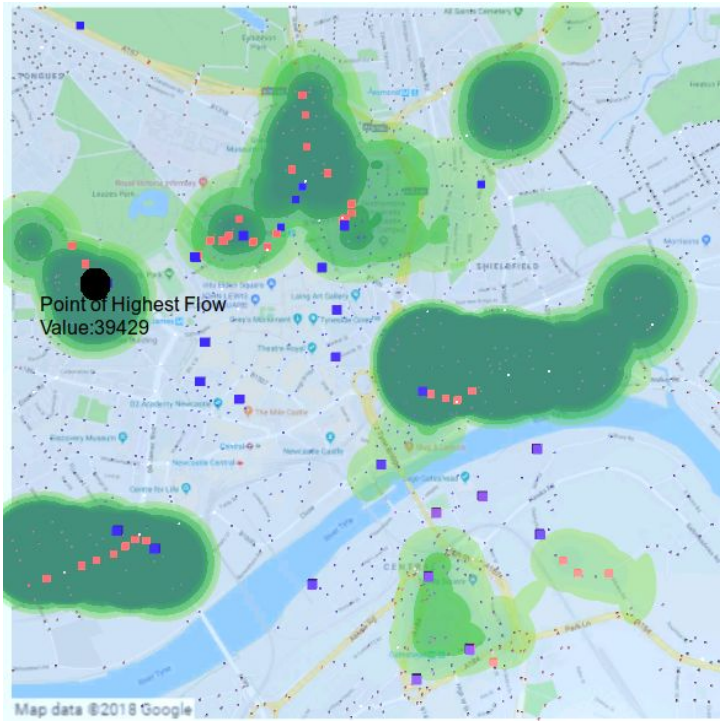
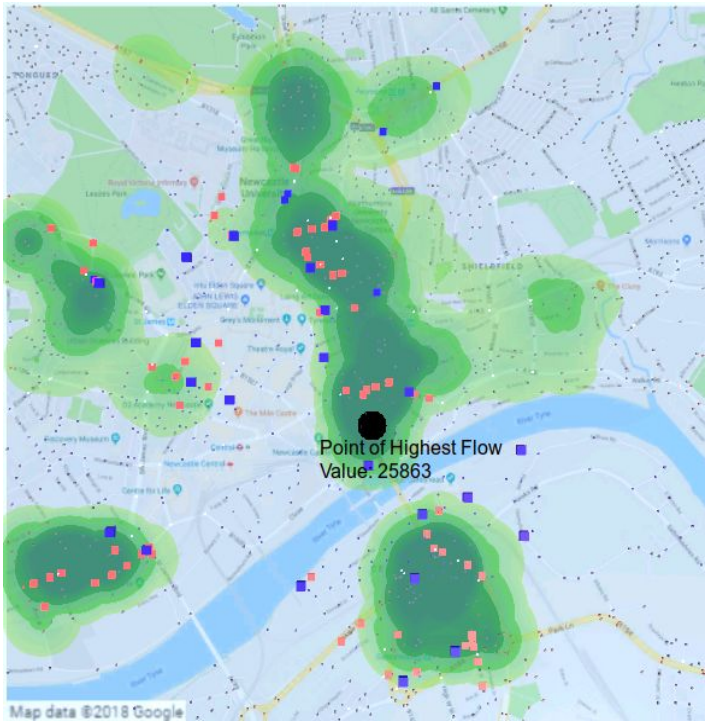


Fig. 4.12

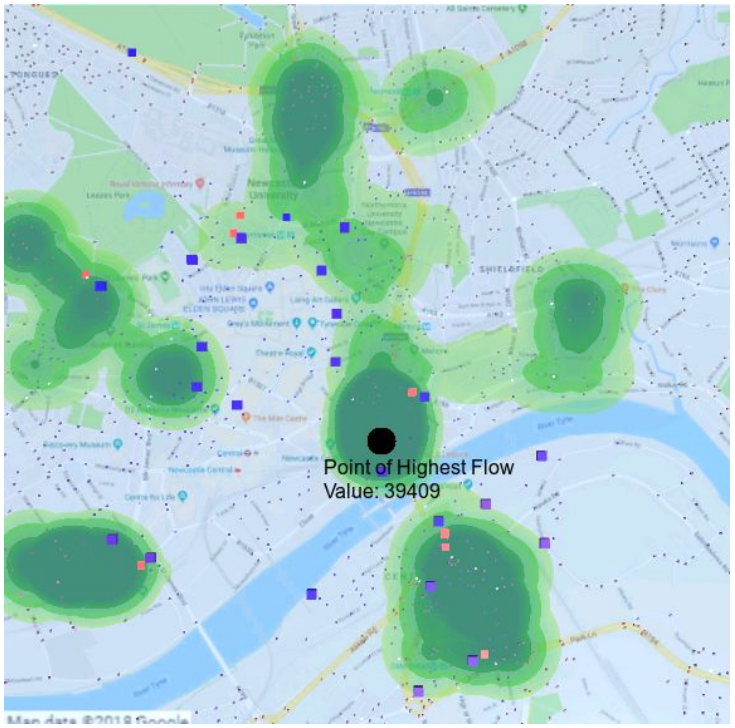
In the heatmaps (Fig. 4.13), flow is most widely distributed across the map for CD. NCD and A* have almost equal values for their peak congestion points, despite the different areas of formed clusters. This is expected since both algorithms use non-collaborative pathing, which does not change to avoid traffic. The choke points created are consistent with the results from the low traffic experiment; the peak congestion point of CD is 35% lower than A* and NCD.



A*



Collaborative diffusion



Non-collaborative diffusion

Fig. 4.13 Heatmaps

The results for agent travel time and path costs indicate that A* is consistently better at providing the best path in terms of cost and journey time, while NCD is the slowest and most costly algorithm of the three (see Appendix 9 and Appendix 8). Collaborative diffusion, on the other hand, provides on average quick routes in terms of journey time, which match the

ones generated by A*. Since the costs of these paths are higher, it is safe to assume that CD guides agents around traffic and sacrifices the cost of the path for an uncongested route.

The higher cost of the paths is a surprising result since NCD is expected to take a more direct path than CD. This most likely indicates a small error in the aggregation or collection of the NCD data, rather than any conclusions about the algorithm itself. However, the importance of agent interaction in CD is further verified by the fact that NCD has the slowest journey times. Without the interaction, the agents do not predict congested routes and increase their journey times.

Although this behaviour leads to more distributed clusters of traffic (as seen in the heatmap), it is still slower than the direct and “brute” solution of A* pathfinding.

Table 4.3

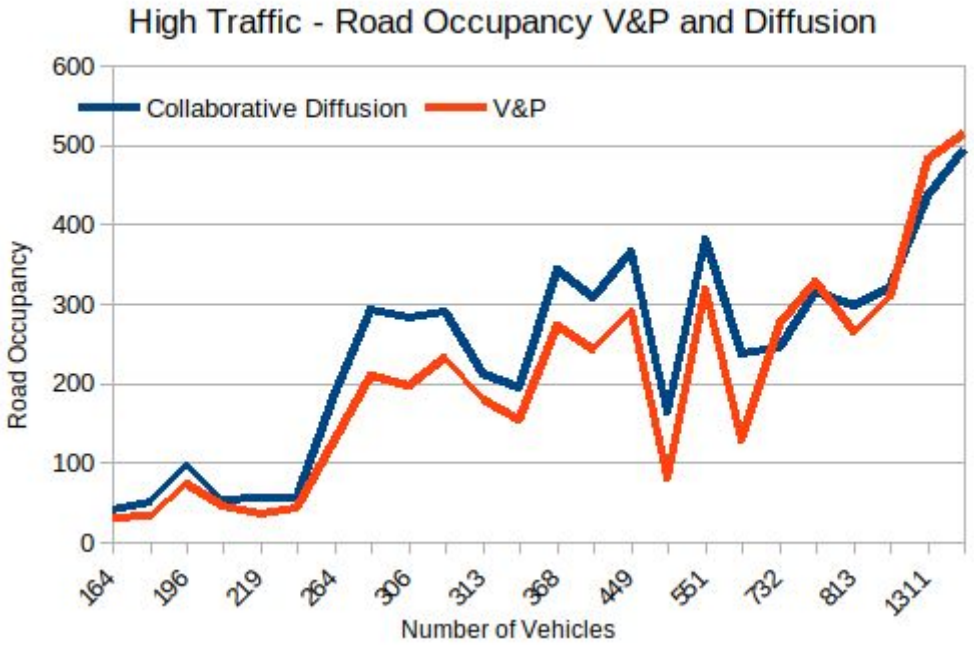
Averages of	A*	Non-collaborative Diffusion (NCD)	Collaborative Diffusion (CD)
Journey Time (seconds):	353.17	447.25	371.8
Path Cost:	141.7	220.56	206.03

V&P and diffusion

A comparison is also made between CD and the proposed solution: V&P. The new algorithm is expected to give better results than diffusion due to the fixed pathfinding behaviour and the addition of the mass flow to the pathfinding.

In the data it is clear that V&P fixes the problem of wandering agents and improves the distribution of flow, while simultaneously reducing the overall flow and road occupancy (Fig. 4.14). The result of this is that V&P has both the lowest amount of flow for all algorithms and also the lowest peak point of congestion (heatmap for V&P, Figure 4.15). It also substantially decreases the road occupancy.

These results indicate that the bottleneck problem discussed in the Design and Implementation section does indeed affect the collaborative diffusion algorithm to a significant extent. The addition of the flow factor in the diffusion calculation successfully distributes traffic more evenly across the map.



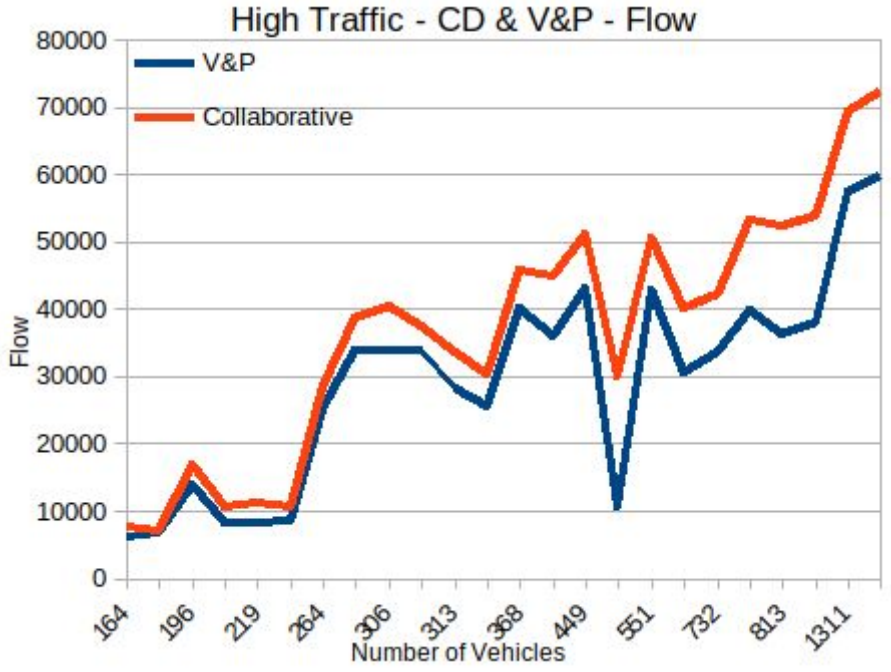


Fig. 4.14

Averages of	A*	Non-collaborative Diffusion (NCD)	Collaborative Diffusion (CD)	V&P
Road Occupancy:	207.2	157.7	238.2	202.3
Flow:	34,208	32,300	37,098	29,170

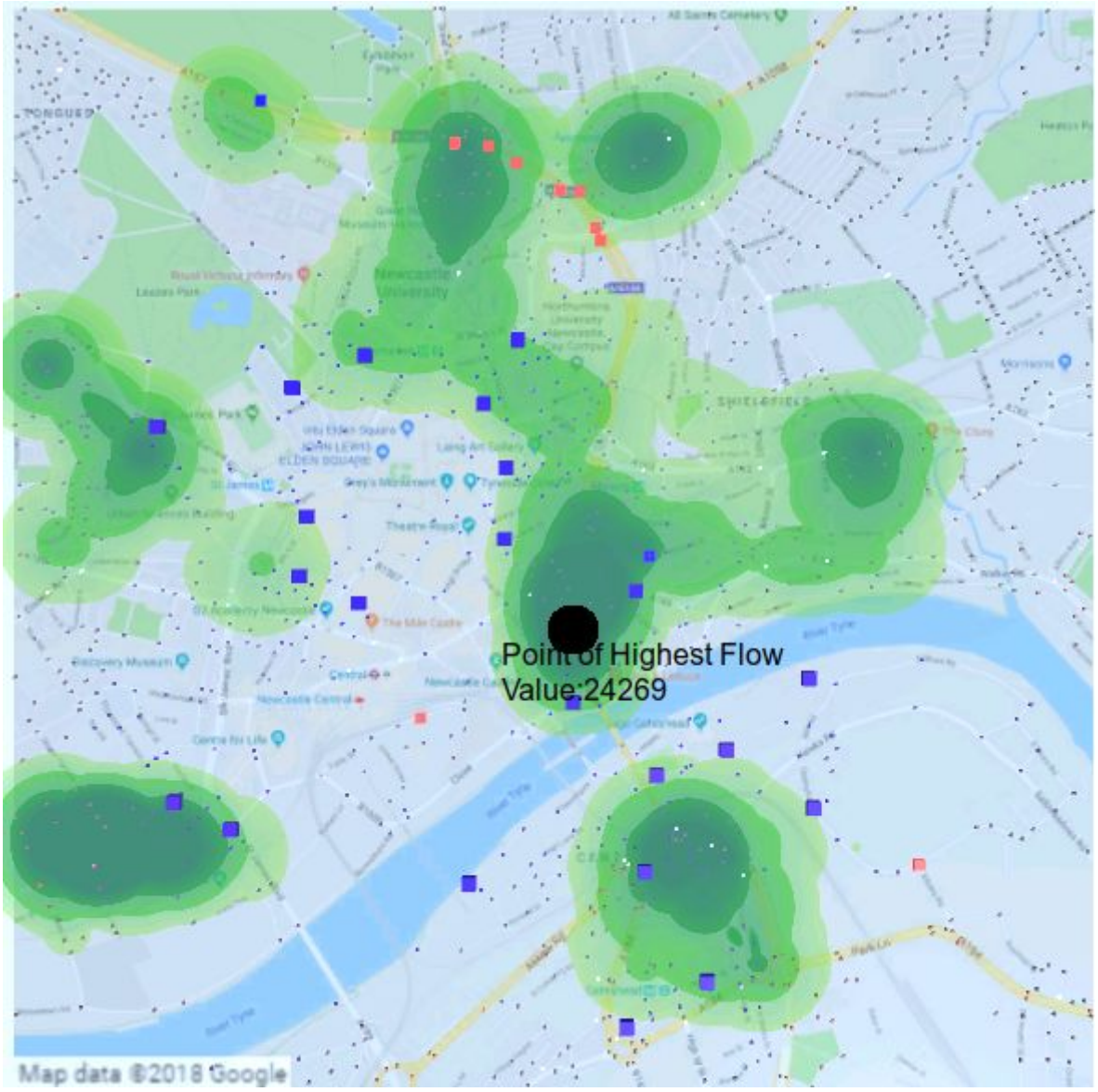


Fig. 4.15 Heatmap for V&P

Average agent travel time and path costs are also substantially decreased in comparison to collaborative diffusion (see Appendix 10). V&P is consistently shown to give the best results for journey times for all algorithms. The path cost is, as expected, more than the cost of A*, as the algorithm sacrifices a direct path for a faster, less-congested path.

Compared to A* there is a 25% increase in the cost of the path and a 5% decrease in the average journey time. The results for agent travel time and path cost support the notion that V&P reduces traffic and improves the collaborative diffusion algorithm.

Table 4.4

Averages of	A*	Non-collaborative Diffusion	Collaborative Diffusion (CD)	V&P

		(NCD)		
Journey Time (seconds):	353.17	447.25	371.8	338.5
Path Cost:	141.7	220.56	206.03	180

4.4 Summary

Through the experiments, valuable insights were obtained for both the Newcastle simulation and the algorithms. The simulation proved to be valid as per its original aim; it captures congestion on a mid-fidelity level. Overall, the data suggests that collaborative diffusion and V&P successfully re-distribute traffic on the map.

For the simulation, all tested metrics were proven to be affected by congestion to some degree in experiment one. As the traffic increased, the congestion metrics increased as well. However, there were a few inconsistencies. Firstly, the journey time distribution was off by a margin in both the high and low groups and it did not seem to correlate with vehicle count or flow as it does in the Urban Observatory data. This was in particular seen in the low traffic groups. Despite this, the average journey times generally trended upwards with increases in vehicles and they remained a good indicator of the collective journey time.

Road occupancy was also behaving slightly inconsistently with regard to the distribution of traffic (seen in the heatmaps). In both experiments, the algorithm with the lowest road occupancy was NCD, but the resulting peak congestion point for NCD in both scenarios was the highest of all algorithms. The clusters generated from flow also showed that NCD agents conformed more to the same roads, which should have produced more road occupancy. The clusters and the peak congestion point results suggest that agents were taking the same roads and grouping together, hence the road occupancy should have been higher. This could also potentially be attributed to the topology of the grid network, where agents are streamlined towards their goal, but this is more unlikely. Despite these errors, the overall behaviour of the simulation correctly captures congestion with regards to its original aim.

In the second and third experiments, it was seen that A* remains the most viable choice for lower traffic and V&P is a faster solution for higher traffic. Collaborative diffusion seemed to redistribute traffic but did not improve the average journey times of agents (compared with A*). This redistribution of traffic could have potentially contributed more to congestion rather than alleviating it. In the high traffic scenario CD had the most traffic flow, which indicates that agents stayed on the map and moved the most. This was attributed to agents potentially wandering around the map looking for a path with less congestion. The movement also potentially generated the high road occupancy values seen in CD for both experiments. However, this notion is contradicted by the fact that it has more distributed traffic than A* and NCD. The travel times for the high traffic scenario for CD were also significantly lower than NCD. This suggests two things:

A. CD reduced congestion by spreading traffic across the map. This increased the overall flow and decreased the concentration of traffic clusters.

B. Agent interaction played a big part, since NCD represents the case of CD with no interaction. CD successfully avoids traffic.

This evasion of traffic, however, caused agents to wander around the map when there were no corridors free of traffic. There was also the potential problem of diffusion bottlenecks that restrict the optimisation of road networks. Due to these behaviours, A* remained the faster solution. V&P successfully optimised CD and reliably proved to be a faster pathfinding algorithm for higher traffic scenarios than A*. It solved both the wandering and bottlenecking behaviour by guiding the agents towards the “flow” of a goal. It improved A*’s journey time by 5%, while at the same time reducing the overall flow, peak congestion and road occupancy. The positive effect of V&P will be exaggerated as traffic increases due to the exponential penalty of built up congestion for non-collaborative algorithms.

The computational statistics of the algorithms were deemed to be outside the scope of the project. The new V&P algorithm was created to be highly parallelisable and should have comparable results to CD. Pathfinding towards multiple goals was a problem with the presented collaborative algorithms, however. The solution to this was to intertwine diffusion patterns so as to direct agents towards specific goals. However, this would not be a viable solution for increasing amounts of new agents with individual goals, as diffusion values need to be calculated and propagated. This remains to a large extent a restriction on the diffusion algorithms. They are instead best employed in scenarios where agents share common goals.

This section provided an in-depth overview of the generated simulation data and algorithms used. The overall results show that collaborative diffusion redistributes traffic and that V&P improves and guides this behaviour to a greater degree.

Chapter 5: Conclusion

This section presents a conclusion and reflection based on the aims and objectives outlined in Chapter 1 (1.3 Project Aims and Objectives). An overview of the original objectives is provided, alongside comments on the achievement of each objective. Possible future work and research is also discussed.

5.1 Satisfaction of Aims and Objectives

Objective	Outcome	Explanation
Research Algorithms and Smart Parking	Achieved	The background research into smart parking and PGI systems provided a comprehensive understanding of the field. An overview of existing navigation methods was provided, with an emphasis on collaborative pathfinding. The potential flaws of direct approaches (i.e. non-collaborative solutions) were discussed and commented on. In-depth research was carried out into the collaborative diffusion algorithm and diffusion as a physical property.
Simulate Congestion – Testing Environment	Partially achieved	The initially-provided environment in the pathfinding framework was deemed generally sufficient. An automated testing structure was built and data extraction was implemented. However, this should have been further developed into a full testing suite for the pathfinding framework, rather than the more manual scenario testing.
Simulate Congestion – Simulation Environment	Achieved	The original aim of creating a mid-fidelity simulation that captured congestion was successfully met. The end result was a fully functional PGI simulation that reacts to traffic. Despite minor errors, the simulation model proved to be an overall valid representation of a road network with regards to the implemented metrics.
Evaluation and Statistics Tools	Achieved	The resulting algorithms and simulation were thoroughly assessed using the created evaluation tools. Using standardised metrics, collaborative diffusion was found to distribute traffic, but it did not effectively reduce the journey times of vehicles. This could be further expanded on by testing different scenarios with higher amounts of traffic.
Algorithm Augmentation	Achieved	In-depth research was carried out into multiple science domains for solutions to the proposed problem. The presented bottleneck issue was successfully resolved and two additional augmentations were produced: collaborative diffusion-correction and V&P. V&P was further

		evaluated as an improvement on the initial collaborative diffusion algorithm.

Overall conclusion

The original aim of the project was to research collaborative diffusion pathfinding and its effects on congestion. This was met with positive results, suggesting that it may in fact be a viable navigation algorithm. The potential improvement of correction or V&P were also suggested. These types of algorithms were found to work best in situations with shared goals between agents, rather than exclusive goals. The developed simulation and experiments serve as proof of concept and a basis for future research.

5.2 What Went Well

There are two main aspects of the project development that went well. The initial structure of the project did not include the theoretical section discussing algorithms. The problem was that this section itself could have been a separate research project. The original structure was changed so as to embed the small research project within the larger one. This was able to successfully remain within the context of the main research area of congestion control.

The research to understand and propose solutions also provided a successful result. A significant amount of time and effort was spent on researching diffusion and fluid dynamics without progressing the solution. Ultimately, however, this cross-domain research turned out to be useful and relevant.

5.3 Improvements

If the project were repeated, I would change the following actions:

- A more concise approach should have been taken when describing results and the developed algorithms, with the aim of the paper being slightly shorter.
- The testing environment and the overall “testing” aspect of the project is below standard and could be significantly improved. More effort should have been spent on developing the testing environment and testing tools.
- The time management of the project could have been slightly better. This was an issue mainly because the bottleneck problem and solution were discovered relatively close to the deadline. In hindsight, more thorough research should have been conducted before starting development.
- In the simulation some of the standard traffic metrics were error-prone. These errors should have been more thoroughly examined and fixed, so as to give more representative results.
- The evaluation of the project should have included how the standard metrics reacted to congestion in the real-life data, and then compared this to how the simulation reacted. This would have further proven the validity of the model.
- The agile approach was not the best methodology to use in this project. A better solution would have been to first evaluate all improvements on the pathfinding framework and then use the “waterfall” methodology to plan ahead. Doing this would have potentially highlighted the bottleneck issue earlier on in development.

5.4 Future Work

This system provides a simulation of a Newcastle PGI network and the analysis of collaborative diffusion as a means of congestion control. However, due to a limited amount of time the solution was not generalised to different cities or scenarios. An easy to implement improvement would be to generate the road networks, parking lots and traffic lights using the OSM database. OSM provides a node-based representation of roads and cities, which could be directly imported to the node-grid system used in the project. This contains a rich dataset that could also be used to improve the accuracy simulation.

The proposed bottleneck property of diffusion could also further be formally proven with the goal of better understanding its effects on pathfinding. The property is most likely a well-researched area in graph theory. The proposed solutions in this paper should also be formally verified and simplified. The resulting V&P solution seems to be an improvement of the collaborative diffusion algorithm, but it should be more thoroughly researched.

Another problem to be solved in future work is an alternative to intertwining goals. The cost of the algorithm as it is currently implemented grows exponentially as the number of exclusive goals increase.

5.5 What Was Learnt

The most important skill learnt from writing this paper was to analyse problems and seek solutions from science. During research and development of the V&P (and other failed solutions before it), sources from different research fields were looked at and then combined into one solution. This newly gained skill serves as a testimony to the utility of the Newcastle Bachelor of Science degree!

References

[1] Arnott, R., Rave, T. and Schob, R., 2005. Alleviating Urban Traffic Congestion. Massachusetts: The MIT Press.
[2] Kodransky, M., Hermann, G., 2011. Europe's Parking U-Turn: From Accommodation to Regulation from <https://itdpdotorg.wpengine.com/wp-content/uploads/2014/07/Europes_Parking_U-Turn_ITDP.pdf>.
[3] Parkhurst, G., 1995. Park and Ride: Could it Lead to an Increase in Car Traffic? Transport Policy, 2(1), pp.15-23.
[4] Yan, G., Yang, W., Rawat, D. and Olariu, S., 2011. SmartParking: A Secure and Intelligent Parking System. IEEE Intelligent Transportation Systems Magazine, 3(1), pp.18-30.
[5] Wang, H. and He, W., 2011. A Reservation-based Smart Parking System. 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).
[6] Geng, Y. and Cassandras, C., 2011. A New Smart Parking System Based on Optimal Resource Allocation and Reservations. 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), pp.979-984.
[7] Pawłowicz, B., Salach, M. and Trybus, B., 2018. Smart City Traffic Monitoring System Based on 5G Cellular Network, RFID and Machine Learning. Advances in Intelligent Systems and Computing, pp.151-165.
[8] Repenning, A., 2006. Collaborative Diffusion. Companion to the 21st ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications - OOPSLA '06.
[9] Muir, D., 1966. Bulk Flow and Diffusion in the Airways of the Lung. British Journal of Diseases of the Chest, 60(4), pp.169-176.
[10] Conrad, R. and Figliozzi, M., 2010. Algorithms to Quantify Impact of Congestion on Time-dependent Real-world Urban Freight Distribution Networks. Transportation Research Record: Journal of the Transportation Research Board, 2168(1), pp.104-113.
[11] Newcastle.urbanobservatory.ac.uk. 2020. Urban Observatory. [online] Available at: <https://newcastle.urbanobservatory.ac.uk/api_docs/> [Accessed 03 May 2020].
[12] 2020. Local Authority Parking Finances in England. RAC Foundation. Available at: <https://www.racfoundation.org/wp-content/uploads/Local_Authority_Parking_Finances_England_2017-18-Leibling_Final.pdf> [Accessed 03 May 2020].
[13] Lin, T., Rivano, H. and Le Mouel, F., 2017. A Survey of Smart Parking Solutions. IEEE Transactions on Intelligent Transportation Systems, 18(12), pp.3229-3253.
[14] Lu, R., Lin, X., Zhu, H. and Shen, X., 2009. SPARK: A New VANET-Based Smart Parking Scheme for Large Parking Lots. IEEE INFOCOM 2009 - The 28th Conference on Computer Communications.
[15] Sharma, P. and Khurana, N., 1970. Study of Optimal Path Finding Techniques. International Journal of Advancements in Technology, 2013, pp.124-130.
[16] Khantanapoka, K. and Chinnasarn, K., 2009. Pathfinding of 2D & 3D Game Real-time Strategy with Depth Direction A* Algorithm for Multi-layer. 2009 Eighth International Symposium on Natural Language Processing.

[17] Hopcroft, J., Schwartz, J. and Sharir, M., 1984. On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE-Hardness of the "Warehouseman's Problem." The International Journal of Robotics Research, 3(4), pp.76-88.
[18] Ko-Hsin, C. W., and Adi, B., 2008. Fast and Memory-efficient Multi-agent Pathfinding. In Proceedings of the Eighteenth International Conference on International Conference on Automated Planning and Scheduling (ICAPS'08). AAAI Press, pp.380-387.
[19] Silver, D., 2005. Cooperative Pathfinding. Proceedings of the 1st Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2005, pp.117-122.
[20] Rhodes, C., Blewitt, W., Sharp, C., Ushaw, G. and Morgan, G., 2014. Smart Routing: A Novel Application of Collaborative Path-Finding to Smart Parking Systems. 2014 IEEE 16th Conference on Business Informatics.
[21] McMillan, C., Hart, E. and Chalmers, K., 2015. Collaborative Diffusion on the GPU for Path-finding in Games. Applications of Evolutionary Computation, pp.418-429.
[22] Harris, M. J., Coombe, G, Scheuermann, T. and Lastra, A., 2002. Physically-based Visual Simulation on Graphics Hardware. In Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware (HWWS '02). Eurographics Association, Goslar, DEU, pp.109-118.
[23] Toffoli, T. and Margolus, N., 1993. Cellular Automata Machines. Cambridge, Mass.: MIT Press.
[24] Ferre, E. and Laumond, J., 2008. A Path Planner for PLM Applications.
[25] Thanou, D., Dong, X., Kressner, D. and Frossard, P., 2017. Learning Heat Diffusion Graphs. IEEE Transactions on Signal and Information Processing over Networks, 3(3), pp.484-499.
[26] Zhang, X., Wang, Z., Ma, C. and Duan, N., 2015. A Diffusion Model with Constant Source and Sinks for Social Graph Partitioning. 2015 IEEE International Conference on Web Services.
[27] https://cao.chem.ufl.edu/. 2020. Diffusion and Fluid Flow. [online] Available at: <https://cao.chem.ufl.edu/wp-content/uploads/sites/22/2015/01/Lecture12-20152.pdf> [Accessed 03 May 2020].
[28] Grady, L. and Polimeni, J., 2010. Discrete Calculus. London: Springer, pp.82-100.
[29] Crank, J., 1976. The Mathematics of Diffusion. Oxford: Clarendon Press.
[30] Radmanesh, M., Kumar, M. and French, D., 2020. Partial Differential Equation-Based Trajectory Planning for Multiple Unmanned Air Vehicles in Dynamic and Uncertain Environments. Journal of Dynamic Systems, Measurement, and Control, 142(4).
[31] Www3.nd.edu. 2020. The Jacobi and Gauss-Seidel Iterative Methods. [online] Available at: <https://www3.nd.edu/~zxu2/acms40390F12/Lec-7.3.pdf> [Accessed 18 May 2020].
[32] NVIDIA Developer. 2020. Chapter 38. Fast Fluid Dynamics Simulation on the GPU. [online] Available at: <https://developer.nvidia.com/gpugems/gpugems/part-vi-beyond-triangles/chapter-38-fast-fluid-dynamics-simulation-gpu > [Accessed 03 May 2020].
[33] Theory.stanford.edu. 2020. Implementation Notes. [online] Available at: <http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html> [Accessed 02 May 2020].
[34] Jansen, M. R. and Sturtevant, N. R., 2008. Direction Maps for Cooperative Pathfinding. In Proceedings of the Fourth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'08). AAAI Press, pp.185-190.
[35] Leif Node. 2020. Flow Field Pathfinding. [online] Available at: <https://leifnode.com/2013/12/flow-field-pathfinding/ > [Accessed 18 May 2020].

[36] Stewart, J., 2012. Calculus. Belmont, Calif.: Brooks/Cole-Cengage Learning.
[37] Pages.mtu.edu. 2020. Mass Transport Processes. [online] Available at: <https://pages.mtu.edu/~reh/courses/ce251/251_notes_dir/node4.html> [Accessed 18 May 2020].
[38] Lozes, F., Elmoataz, A. and Lezoray, O., 2014. Partial Difference Operators on Weighted Graphs for Image Processing on Surfaces and Point Clouds. IEEE Transactions on Image Processing, 23(9), pp.3896-3909.
[39] Eager, D., Pendrill, A. and Reistad, N., 2020. Beyond Velocity and Acceleration: Jerk, Snap and Higher Derivatives.
[40] Miyazaki, R., Yoshida, S., Dobashi, Y. and Nishita, T., n.d. A Method for Modeling Clouds Based on Atmospheric Fluid Dynamics. Proceedings Ninth Pacific Conference on Computer Graphics and Applications. Pacific Graphics 2001.
[41] Sfu.ca. 2020. PHYS 101 Lecture 29 - Viscosity. [online] Available at: <http://www.sfu.ca/~boal/101lecs/101lec29.pdf > [Accessed 03 May 2020].
[42] Bickel, P., Chen, C., Kwon, J., Rice, J., van Zwet, E. and Varaiya, P., 2007. Measuring Traffic. Statistical Science, 22(4), pp.581-597.
[43] Netraveldata.co.uk. 2020. Tyne and Wear Open Data Services Platform API Specification. [online] Available at: <https://www.netraveldata.co.uk/wp-content/uploads/2015/11/333551-OpenDataService-API Specification.pdf> [Accessed 02 May 2020].
[44] Covid.view.urbanobservatory.ac.uk. 2020. Effect of Virus Measures | Urban Observatory. [online] Available at: <https://covid.view.urbanobservatory.ac.uk/#traffic-summary> [Accessed 18 May 2020].
[45] Kobus, M., Gutiérrez-Puigarnau, E., Rietveld, P. and van Ommeren, J., 2013. The On-street Parking Premium and Car Drivers' Choice between Street and Garage Parking. Regional Science and Urban Economics, 43(2), pp.395-403.
[46] Ibeas, A., dell'Olio, L., Bordagaray, M. and Ortúzar, J., 2014. Modelling Parking Choices Considering User Heterogeneity. Transportation Research Part A: Policy and Practice, 70, pp.41-49.
[47] Overpass-turbo.eu. 2020. Overpass Turbo. [online] Available at: <https://overpass-turbo.eu/> [Accessed 01 May 2020].

Appendices

Appendix 1

y = 1, difCoef = 0.4, decayRate = 0.1, EvasionStrength = 2

Node 2, t1 = 0.9×(0 + 0.4×(10−0 + 0−0 + 0−0)) = 3.6
Node 3, t1 = 0.9×(0 + 0.4×(0−0 + 0−0)) = 0
Node 4, t1 = 0
Node 5, t1 = 0
Node 6, t1 = 0
Node 7, t1 = 0
Node 8, t1 = 0

Node 2, t2 = 0.9×(3.6 + 0.4×(10−3.6 + 0−3.6 + 0−3.6)) = 2.952
Node 5, t2 = 0.9×(0 + 0.4×(3.6)) = 1.3
Node 3, t2 = 0.9×(0 + 0.4×(3.6)) = 1.3
Node 4, t2 = 0
Node 5, t2 = 0
Node 6, t2 = 0
Node 7, t2 = 0
Node 8, t2 = 0

Node 2, t3 = 0.9×(3 + 0.4×(10−3 + 1.3−3 + 1.3−3)) = 3.996
Node 3, t3 = 0.9×(1.3 + 0.4×(3−1.3 + 0−1.3 + 0−1.3)) = 0.846
Node 4, t3 = 0.9×(0 + 0.4×(1.3)) = 0.468
Node 5, t3 = 0.9×(1.3 + 0.4×(3−1.3 + 0−1.3)) = 1.314
Node 6, t3 = 0.9×(0 + 0.4×(1.3)) = 0.468
Node 7, t3 = 0
Node 8, t3 = 0

Node 2, t4 = 0.9×(4 + 0.4×(10−4 + 1.3−4 + 0.8−4)) = 3.636
Node 3, t4 = 0.9×(0.8 + 0.4×(4−0.8 + 0.5−0.8)) = 1.764
Node 4, t4 = 0.9×(0.5 + 0.4×(0.8−0.5 + 0−0.5)) = 0.378
Node 5, t4 = 0.9×(1.3 +0.4×(4−1.3 + 0.5−1.3)) = 1.854
Node 6, t4 = 0.9×(0.5 + 0.4×(1.3−0.5 + 0−0.5)) = 0.558
Node 7, t4 = 0.9×(0 + 0.4×(0.5)) = 0.18
Node 8, t4 = 0.9×(0 + 0.4×(0.5)) = 0.18

Node 2, t5 = 0.9×(3.6 + 0.4×(10−3.6 + 1.8−3.6 + 1.9−3.6)) = 4.284
Node 3, t5 = 0.9×(1.8 + 0.4×(3.6−1.8 + 0.4−1.8)) = 1.764
Node 4, t5 = 0.9×(0.4 + 0.4×(1.8−0.4 + 0.2−0.4)) = 0.792
Node 5, t5 = 0.9×(1.9 +0.4×(3.6−1.9 + 0.6−1.9)) = 1.854
Node 6, t5 = 0.9×(0.6 + 0.4×(1.9−0.6 + 0.2−0.6)) = 0.864
Node 7, t5 = 0.9×(0.2 + 0.4×(0.6-0.2 + 0.2-0.2)) = 0.324
Node 8, t5 = 0.9×(0.2 + 0.4×(0.4-0.2 + 0.2-0.2)) = 0.252

Appendix 2

771
731
691
651
611
571
531
491
451
452
453
454
455
456
457
458
498
538
578
618
658
698
738
778
772
773
774
775
776
777
732
733
734
735
736
737
692
612
652
572
532
492
493
494
495
496
497
697
656
616
615
577
574
533
534
535
536
537
613
654
695
694
693
653
end=573
start=695

Appendix 3

diffusion = diffusion / EvasionStrength * Number of Avatars Overlapping
y = 1, difCoef = 0.4, decayRate = 0.1, EvasionStrength = 2

Node 2, t1 = 4.3/(5*2) = 0.43
Node 3, t1 = 1.8/(5*2) = 0.18
Node 4, t1 = 0.8/(5*2) = 0.08

Node 2, t2 = (0.9(0.43 + 0.4(10 - 0.43 + 0.18 - 0.43 + 1.9 - 0.43)))/10 = 0.4271
Node 3, t2 = (0.9(0.18 + 0.4(0.08 - 0.18 + 0.43 - 0.18)))/10 = 0.0216
Node 4, t2 = (0.9(0.08 + 0.4(0.18 - 0.08 + 0.25 - 0.08)))/10 = 0.01692
Node 5, t2 = 0.9(1.9 + 0.4(0.43 - 1.9 + 0.9 - 1.9)) = 0.8208
Node 6, t2 = 0.9(0.9 + 0.4(0.32 - 0.9 + 1.9 - 0.9)) = 0.9612
Node 7, t2 = 0.9(0.32 + 0.4(0.9 - 0.32 + 0.25 - 0.32)) = 0.4716
Node 8, t2 = 0.9(0.25 + 0.4(0.32 - 0.25 + 0.08 - 0.25)) = 0.189

Node 2, t3 = (0.9(0.4271 + 0.4(10 - 0.4271 + 0.0216 - 0.4271 + 0.8208 - 0.4271)))/10 = 0.382
Node 3, t3 = (0.9(0.0216 + 0.4(0.01692 - 0.0216 + 0.4271 - 0.0216)))/10 = 0.016
Node 4, t3 = (0.9(0.01692 + 0.4(0.0216 - 0.01692 + 0.189 - 0.01692)))/10 = 0.008
Node 5, t3 = 0.9(0.8208 + 0.4(0.9612 - 0.8208 + 0.4271 - 0.8208)) = 0.647
Node 6, t3 = 0.9(0.9612 + 0.4(0.4716 - 0.9612 + 0.8208 - 0.9612)) = 0.638
Node 7, t3 = 0.9(0.4716 + 0.4(0.9612 - 0.4716 + 0.189 - 0.4716)) = 0.500
Node 8, t3 = 0.9(0.189 + 0.4(0.4716 - 0.189 + 0.01692 - 0.189)) = 0.210

Node 2, t4 = 0.9(0.382 + 0.4(10 - 0.382 + 0.016 - 0.382 + 0.647 - 0.382))/10 = 0.377
Node 3, t4 = 0.9(0.016 + 0.4(0.008 - 0.016 + 0.382 - 0.016))/10 = 0.014
Node 4, t4 = 0.9(0.008 + 0.4(0.016 - 0.008 + 0.210 - 0.008))/10 = 0.008
Node 5, t4 = 0.9(0.647 + 0.4(0.382 - 0.647 + 0.638 - 0.647)) = 0.484
Node 6, t4 = 0.9(0.638 + 0.4(0.500 - 0.638 + 0.647 - 0.638)) = 0.528
Node 7, t4 = 0.9(0.500 + 0.4(0.638 - 0.500 + 0.210 - 0.500)) = 0.395
Node 8, t4 = 0.9(0.210 + 0.4(0.500 - 0.210 + 0.008 - 0.210)) = 0.221

Node 2, t5 = 0.9(0.377 + 0.4(10 - 0.377 + 0.014 - 0.377 + 0.484 - 0.377))/10 = 0.371
Node 3, t5 = 0.9(0.014 + 0.4(0.008 - 0.014 + 0.377 - 0.014))/10 = 0.014
Node 4, t5 = 0.9(0.008 + 0.4(0.014 - 0.008 + 0.221 - 0.008))/10 = 0.009
Node 5, t5 = 0.9(0.484 + 0.4(0.377 - 0.484 + 0.528 - 0.484)) = 0.412
Node 6, t5 = 0.9(0.528 + 0.4(0.395 - 0.528 + 0.484 - 0.528)) = 0.411
Node 7, t5 = 0.9(0.395 + 0.4(0.528 - 0.395 + 0.221 - 0.395)) = 0.340
Node 8, t5 = 0.9(0.221 + 0.4(0.395 - 0.221 + 0.008 - 0.221)) = 0.185

Appendix 4

Node 8, t1 = 0.9(0.185 + 0.4(0.009 - 0.185 + 0.34 - 0.185))/10 = 0.016
Others stay constant or are not affected

Node 8, t2 = 0.9(0.016 + 0.4(0.009 - 0.016 + 0.34 - 0.016))/10 = 0.013
Node 7, t2 = 0.9(0.34 + 0.4(0.412 - 0.34 + 0.016 - 0.34)) = 0.215
Others stay constant

Node 8, t3 = 0.9(0.013 + 0.4(0.009 - 0.013 + 0.215 - 0.013)) = 0.083
Node 7, t3 = 0.9(0.215 + 0.4(0.412 - 0.215 + 0.013 - 0.215))/10 = 0.019
Node 6, t3 = 0.9(0.411 + 0.4(0.215 - 0.411 + 0.412 - 0.411)) = 0.3
Others stay constant

Node 8, t4 = 0.9(0.083 + 0.4(0.009 - 0.083 + 0.019 - 0.083)) = 0.025
Node 7, t4 = 0.9(0.019 + 0.4(0.083 - 0.019 + 0.3 - 0.019))/10 = 0.014
Node 6, t4 = 0.9(0.3 + 0.4(0.019 - 0.3 + 0.412 - 0.3)) = 0.21
Node 5, t4 = 0.9(0.412 + 0.4(0.3 - 0.412 + 0.37 - 0.412)) = 0.315

Node 8, t5 = 0.9(0.025 + 0.4(0.009 - 0.025 + 0.014 - 0.025)) = 0.013
Node 7, t5 = 0.9(0.014 + 0.4(0.025 - 0.014 + 0.21 - 0.014)) = 0.087
Node 6, t5 = 0.9(0.21 + 0.4(0.014 - 0.21 + 0.315 - 0.21))/10 = 0.016
Node 5, t5 = 0.9(0.315 + 0.4(0.21 - 0.315 + 0.37 - 0.315)) = 0.266

Node 8, t6 = 0.9(0.013 + 0.4(0.009 - 0.013 + 0.087 - 0.013)) = 0.037
Node 7, t6 = 0.9(0.087 + 0.4(0.016 - 0.087 + 0.013 -0.087)) = 0.026
Node 6, t6 = 0.9(0.016 + 0.4(0.087 - 0.016 + 0.266 - 0.016))/10 = 0.012
Node 5, t6 = 0.9(0.266 + 0.4(0.016 - 0.266 + 0.37 - 0.266)) = 0.187

Node 8, t7 = 0.9(0.037 + 0.4(0.009 - 0.037 + 0.087 - 0.037)) = 0.041
Node 7, t7 = 0.9(0.026 + 0.4(0.012 - 0.026 + 0.037 -0.026)) = 0.022
Node 6, t7 = 0.9(0.012 + 0.4(0.187 - 0.012 + 0.026 - 0.012)) = 0.079
Node 5, t7 = 0.9(0.187 + 0.4(0.012 - 0.187 + 0.37 - 0.187))/10 = 0.017

Node 8, t8 = 0.9(0.041 + 0.4(0.009 - 0.041 + 0.022 - 0.041)) = 0.019
Node 7, t8 = 0.9(0.022 + 0.4(0.079 - 0.022 + 0.041 -0.022)) = 0.047
Node 6, t8 = 0.9(0.079 + 0.4(0.022 - 0.079 + 0.017 - 0.079)) = 0.028
Node 5, t8 = 0.9(0.017 + 0.4(0.079- 0.017 + 0.37 - 0.017))/10 = 0.01647

Node 8, t9 = 0.9(0.019 + 0.4(0.009 - 0.019 + 0.047 - 0.019)) = 0.02358
Node 7, t9 = 0.9(0.047 + 0.4(0.028- 0.047 + 0.019 -0.047)) = 0.02538
Node 6, t9 = 0.9(0.028 + 0.4(0.047 - 0.028 + 0.01647 - 0.028)) = 0.0278892
Node 5, t9 = 0.9(0.01647 + 0.4(0.028 - 0.01647 + 0.37 - 0.01647)) = 0.1462446

Appendix 5

PATH B

Node	Cost
2	0.6152798
5	0
6	0.00608831
7	0.0003454458
8	0.005632102

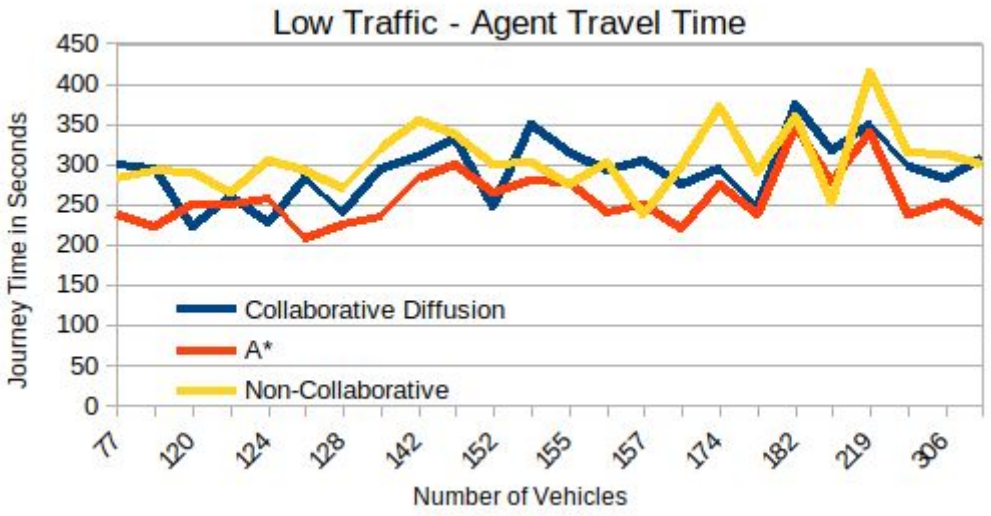
PATH A

Node	Cost
2	0.6152798
3	0.9299771
4	0.05632947

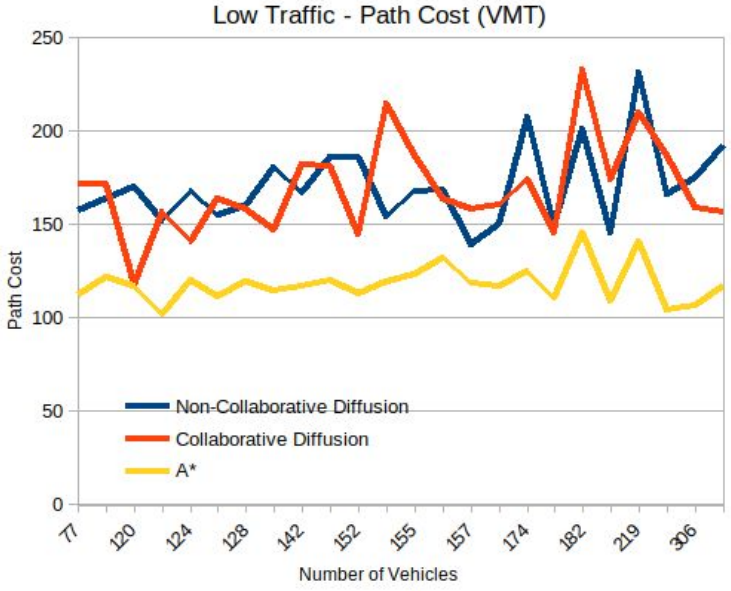
Appendix 6

name	wait_time	time_start	time_end
Work	8	5	8
Shopping	2	13	16
Leisure	3	18	20
Other	2		

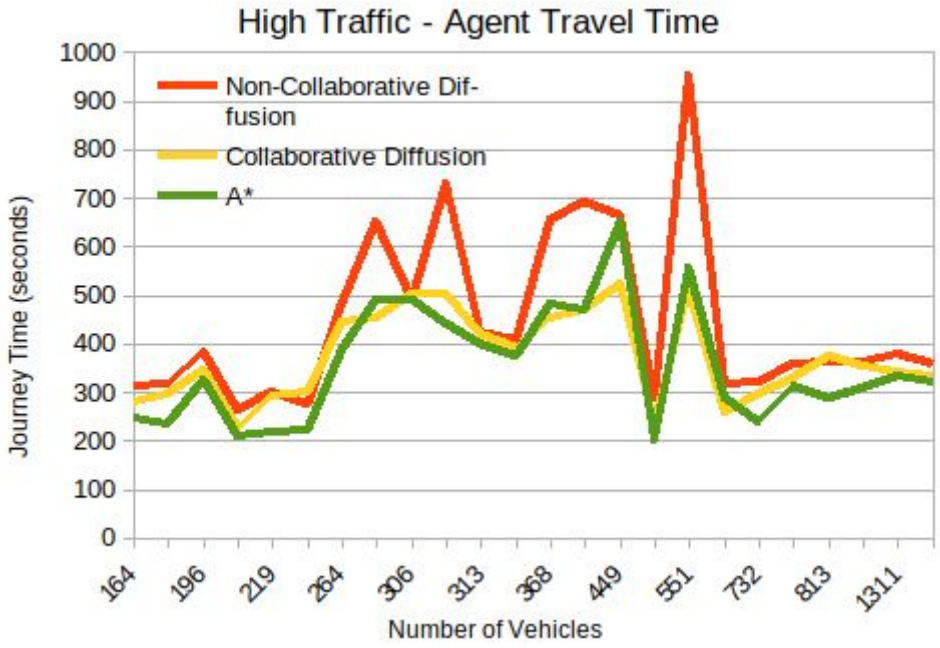
Appendix 7



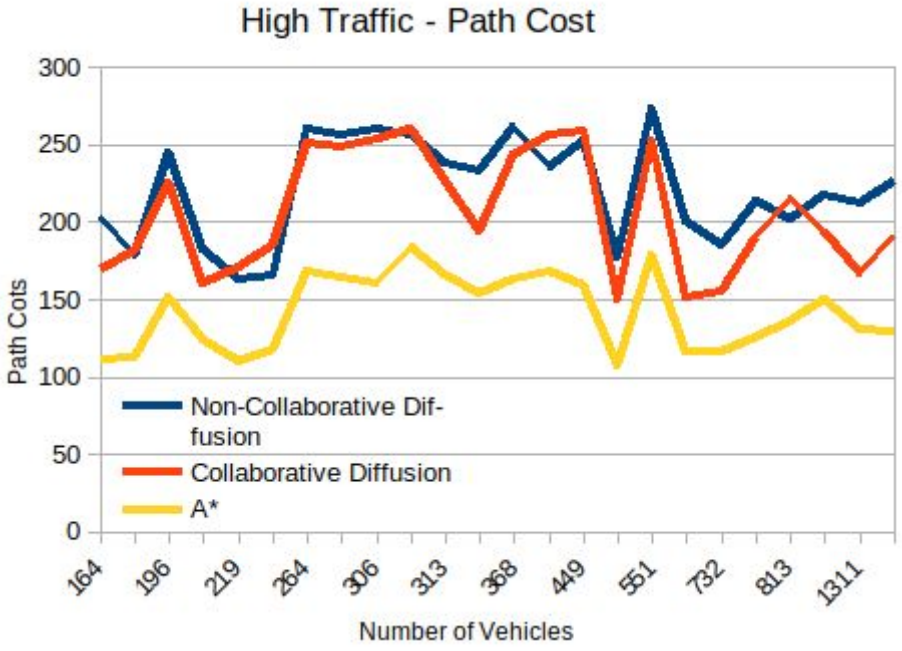
Appendix 8



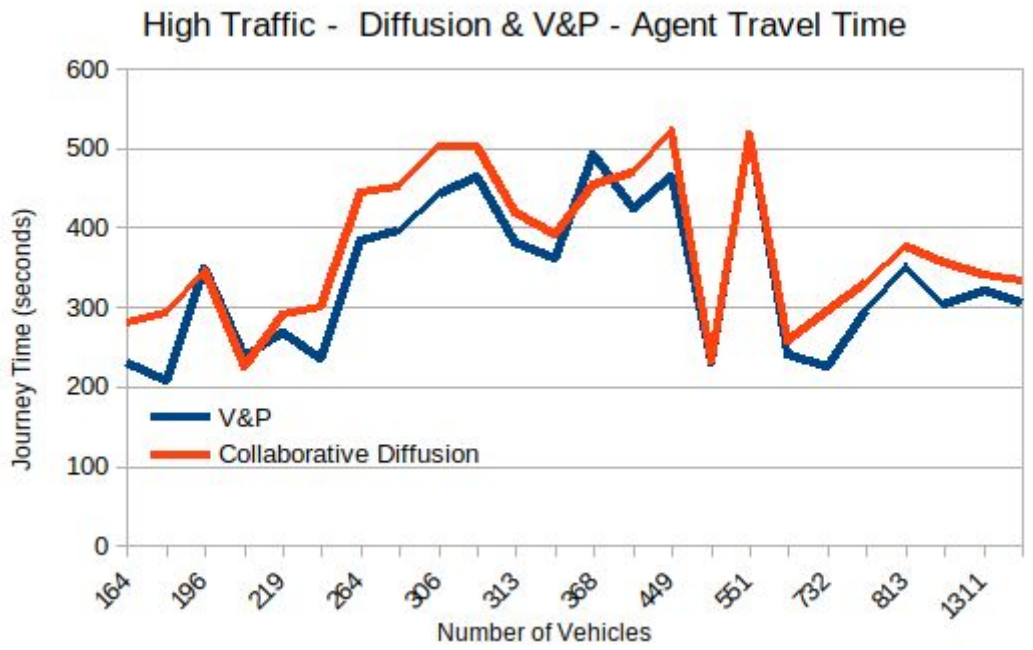
Appendix 9

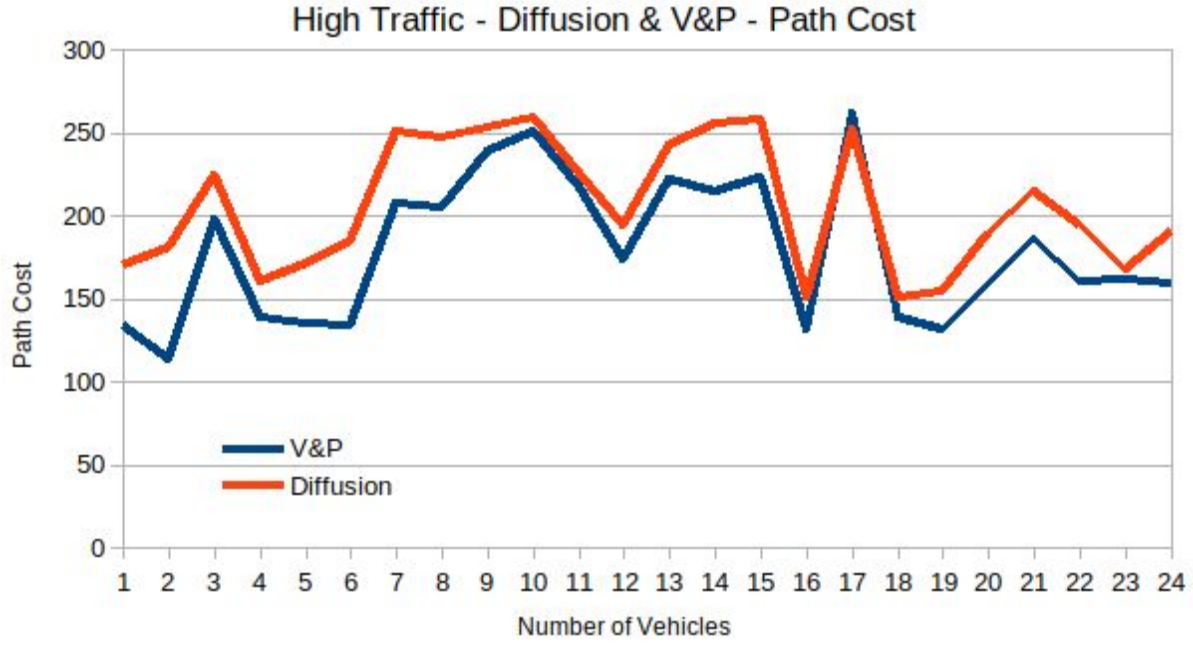


Appendix path cost for high traffic



Appendix 10


High Traffic - Diffusion & V&P - Agent Travel Time
Journey Time (seconds)
600
500
400
300
200
100
0
V&P
Collaborative Diffusion
164
196
219
264
306
313
368
449
551
732
813
1311
Number of Vehicles


High Traffic - Diffusion & V&P - Path Cost
Path Cost
300
250
200
150
100
50
0
V&P
Diffusion
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Number of Vehicles