Mesoscopic modelling of pedestrian movement using Carma and its tools

Mesoscopic modelling of pedestrian movement using Carma and its tools

VASHTI GALPIN, University of Edinburgh
NATALIA ZOŃ, University of Edinburgh
PIA WILSDORF, University of Rostock
STEPHEN GILMORE, University of Edinburgh

In this paper we assess the suitability of the Carma (Collective Adaptive Resource-sharing Markovian Agents) modelling language for mesoscopic modelling of spatially-distributed systems where the desired model lies between an individual-based (microscopic) spatial model and a population-based (macroscopic) spatial model. Our modelling approach is mesoscopic in nature because it does not model the movement of individuals as an agent-based simulation in two-dimensional space, nor does it make a continuous-space approximation of the density of a population of individuals using partial differential equations. The application which we consider is pedestrian movement along paths which are expressed as a directed graph. In the models presented, pedestrians move along path segments at rates which are determined by the presence of other pedestrians, and make their choice of the path segment to cross next at the intersections of paths. Information about the topology of the path network and the topography of the landscape can be expressed as separate functional and spatial aspects of the model by making use of Carma language constructs for representing space. We use simulation to study the impact on the system dynamics of changes to the topology of paths and show how Carma provides suitable modelling language constructs which make it straightforward to change the topology of the paths and other spatial aspects of the model without completely restructuring the Carma model. Our results indicate that it is difficult to predict the effect of changes to the network structure and that even small changes can have significant effects.

CCS Concepts: •**Computing methodologies** →**Model development and analysis; Simulation support systems;** •**Theory of computation** →**Process calculi;** •**Applied computing** →*Law, social and behavioral sciences;*

Additional Key Words and Phrases: stochastic simulation, collective adaptive systems, process algebra

ACM Reference format:
Vashti Galpin, Natalia Zoń, Pia Wilsdorf, and Stephen Gilmore. 2017. Mesoscopic modelling of pedestrian movement using Carma and its tools. *ACM Trans. Model. Comput. Simul.* 1, 1, Article 1 (January 2017), **??** pages.
DOI: 10.1145/3155338

1 INTRODUCTION

The modelling of pedestrian and crowd movement has been an area of research interest for many years in domains as diverse as town planning, parade routing, and emergency egress placement for

This work is supported by the EU project QUANTICOL, 600708.
Author's addresses: V. Galpin, N. Zoń and S. Gilmore, School of Informatics, University of Edinburgh; P. Wilsdorf, Institute of Computer Science, University of Rostock.
 XXXX-XXXX/2017/1-ART1 $$15.00
DOI: 10.1145/3155338

auditoriums and stadia [**? ? ?**]. This paper investigates how the formal language CARMA can be used to express models of pedestrian movement over networks of paths, and takes a *mesoscopic* approach to this spatial modelling rather than *microscopic* or *macroscopic*. The mesoscopic approach is a middle ground between considering a discrete (microscopic) model of individual movement in two-dimensional space or a continuous (macroscopic) model based on partial differential equations (PDEs) where both change over time and change over space are described by continuous quantities. Another approach to modelling the dynamic interaction of populations, such as species of molecules in biology, is the mean field/fluid approximation technique that assumes that populations are well-mixed and makes no spatial distinctions. This would not be useful for the type of pedestrian modelling considered here which is inherently spatial.

In the mesoscopic approach, the model components which represent pedestrians act as individual decision-making entities when they make probabilistic choices about their future movement and function as a collective population mass when we consider how they impede the movement of other pedestrians. Neither of these views of model components (as individual entities or as population mass) is dominant and the model needs to make use of both views to represent how congestion arises along paths and how it impedes the progress of individuals. Together the two views allow us to represent *intelligent density-dependent movement* which captures the behaviour of decision-making individuals adapting to continuously-changing information about the collective to which they belong.

The modelling language which we use, CARMA [**? ?**], has been designed for representing and analysing Collective Adaptive Systems (CAS) which consist of multiple components (or agents) interacting collaboratively on common goals, and competing on individual goals. CAS are characterised by the fact that each component does not have *global* knowledge of the whole system but rather has only *local* information on which to base their decisions and act. CAS are called *collective* because of the interaction of many components, and *adaptive* because they respond to changes in the environment in which they operate. They are often characterised as having emergent behaviour which cannot be predicted in advance by considering the individual components in isolation from each other. The focus on local information as opposed to global knowledge suggests that space often plays an important role in CAS. While this is not necessarily true of all CAS since the distinction between local and global may be logical or virtual rather than physical, it is true for many CAS, and hence this is an important part of understanding their behaviour.

Modelling of CAS is crucial because it is infeasible to understand the behaviour of the overall system solely by inspecting the behaviour of the components. Modelling allows us to experiment with alternative designs of a system at the planning stage before it is realised in a concrete implementation and opened for use. CARMA has been used for modelling particular instances of CAS such as smart transport systems, smart energy grids, and other resource-sharing systems. Not only does CARMA provide a language designed for the styles of interaction required by collective adaptive systems – including attribute-based communication – it also allows for the separation of concerns in models. Components describe the behaviour of agents in the system together with their attributes in a generic manner, so that they are able to move over any spatial structure (as we will show) rather than being fixed to a specific topology. Attributes then allow the modelling of well-structured movement; for example in enforcing constraints which keep the pedestrians on the footpaths, and keep the cyclists on the cycle paths.

In our focus on pedestrian movement along a network of paths, we consider that each pedestrian has only local information about the number of pedestrians on path segments, and they wish to get to their destination as efficiently as possible. An obvious approach to take when presented with the choice of two paths (that both lead to the destination) is to take the one with the least oncoming

pedestrian traffic, although in very crowded situations, considering the pedestrian traffic flow in their current direction of travel may be important as well. The network of paths could be a specific part of a city, a pedestrianised network of lanes, or paths through a large park.

We consider two models in this paper. The first of these is a basic model where there are two groups of pedestrians starting on opposite sides of the network who wish to traverse the paths to get to the other side (opposite to where they started). This scenario could arise in a city where there are two train stations on opposite sides of the central business district serving the eastern and the western suburbs of the city, and a number of people who commute from the west work close to the east station and vice versa. During rush hour in the morning or evening, people want to traverse the park or lanes as efficiently as possible, and we wish to investigate what features of the network of paths help towards this goal. If there are multiple paths, it would seem in advance that it makes sense to use some paths for one direction and other paths for the other direction. This raises the question of what routing information – such as signage – is sufficient for the two groups of pedestrians to separate out onto different paths. Even for this first basic model our experiments demonstrate that this is not as straightforward as one may at first think.

Our second and more realistic pedestrian movement model focusses on an actual park in Edinburgh called the Meadows, and takes into account different directions of pedestrian flows at different times of the day using information inferred from real measurement data collected by the City of Edinburgh Council. In this model we consider the effect of the closure of selected paths (because of maintenance, events, or other reasons) on the pedestrian flow across the Meadows. We present the results of our analysis as two-dimensional density graphs showing the force of flow *in situ* in the spatial network context.

This paper shows how these two models can be expressed in CARMA and specified in CASL, the language of the CARMA Eclipse Plug-In, a fully-featured modelling and analysis environment for CARMA. We first present a brief discussion of the CARMA language before describing the modelling of the scenario in more detail and presentation of our experiments and results from various networks, followed by conclusions and discussion of future work.

The paper makes the following original contributions.

- We present the first example of mesoscopic modelling in CARMA, showing how the language can represent individual probabilistic decisions in a density-dependent context.
- We provide a structured spatial analysis of a real-world scenario in a mesoscopic model which has been parameterised with measurement data from an open data source.

This work extends that published as [**?**]. Our earlier paper presented a basic model similar in style to the one presented here, however, the rate functions that affected choice of path and movement speed which were used there were based on the number of pedestrians at the nodes of the networks rather than the number of pedestrians on the edges of the network. As we will discuss later, this leads to different outcomes. The present paper adds the entirely new Meadows model and shows how empirical measurement data can be used to parameterise a realistic model of a real-world scenario in CARMA.

2 CARMA

CARMA is an expressive process calculus which has been developed specifically for the modelling of collective adaptive systems. A full description of the language can be found in [**? ?**]. Here, we give a brief outline.

A CARMA model consists of a collective N and the environment $\mathcal{E}$ in which it operates, using the syntax N **in** $\mathcal{E}$. A collective is either a component C or collectives in parallel $N \parallel N$. Each component is either null, $\mathbf{0}$, or a combination of behaviour described by a process P and a store of

attributes γ, denoted by (P, γ). We use function notation to denote store access, thus if $\gamma = \{x \mapsto v\}$ then $\gamma(x) = v$. As an example, a (small) collective of two components representing pedestrians travelling in opposite directions could be represented thus in CARMA:

$$(\mathit{Pedestrian}, \{\ \mathit{direction} \mapsto \mathit{west}\ \}) \parallel (\mathit{Pedestrian}, \{\ \mathit{direction} \mapsto \mathit{east}\ \})$$

Prefix (.), constant definitions ($\stackrel{\text{def}}{=}$), choice (+) and parallel composition (|) can be expressed in the standard manner by defining P appropriately. Additionally, there is the **nil** process which does nothing, the **kill** process which results in the component being removed from the collective, and the option of prefixing a process with a predicate $[\pi]P$, in which case the process P can only proceed if the predicate π evaluates to *true* using the values of the attributes in the component's store γ. To improve readability we sometimes parenthesise the process expression P, writing this term as $[\pi](P)$. The meaning is unchanged. A typical use of these constructs would be the definition of a *Pedestrian* component who might decide to have a picnic in the park if the weather is sunny or go home for lunch if the weather is rainy. This is expressed in CARMA thus:

$$\begin{array}{rcl} \mathit{Pedestrian} & \stackrel{\text{def}}{=} & [\mathit{weather} = \mathit{sunny}]\,(\mathsf{goToPark}\langle\rangle.\mathit{HavePicnic}) \\ & + & [\mathit{weather} = \mathit{raining}]\,(\mathsf{goHome}\langle\rangle.\mathit{MakeLunch}) \end{array}$$

Process prefixes provide value-passing unicast and broadcast communication using predicates over the attributes in the stores of both the component which is sending the value and the component which is receiving the value. Communication between components will only take place if the predicates over both stores evaluate to *true*. The value *false* indicates that no communication partner is needed (when broadcasting). Furthermore, attribute values can be updated (probabilistically) on completion of an action.

Unicast communication is blocking; the sender cannot output values unless there is a matching input action which can be performed by another component. In contrast, broadcast is not blocking, and we can use a specific form with a constant *false* predicate to allow components to act without interaction with other components, as seen in the example to follow. The constants *true* and *false* in CARMA models are written as $\top$ and $\bot$ here.

The syntax of a non-blocking broadcast on name α is $\alpha^\star[\pi]\langle\vec{v}\rangle\sigma$ where π is a predicate which must be satisfied by all processes wishing to receive this broadcast. The vector $\vec{v}$ is a vector of values to be communicated; this vector may be empty. The suffix σ is an *update* of variables in the local store of a component. A component refers to an attribute in its own local store by prefixing the name of the attribute with the word 'my' (similar to the use of the keyword 'this' in Java) so an update to store the value of x as the new value of $\mathsf{my}.x$ is written as $\{\mathsf{my}.x \leftarrow x\}$. As an example, the prefix process term $\mathsf{move}_k^\star[\bot]\langle\rangle\{\mathsf{my}.\ell \leftarrow k\}.M$ broadcasts that it is performing a move_k activity, updates its local ℓ values, and continues as the process M.

The environment contains both the global store and an evolution rule which returns a tuple of four functions $(\mu_p, \mu_w, \mu_r, \mu_u)$ known as the *evaluation context*. Communication between sender s and receiver r on activity α has both an associated *probability* (determined by μ_p) and a *weight* (determined by μ_w). These functions depend on activity α and both the attribute values of the sender (in the store γ_s) and the attribute values of the receiver (in the store γ_r). The activity *rate* however depends on only the attribute values in the store of the sender (γ_s); the attribute values of the receiver do not affect the rate at which a communication activity is performed. An example rate function in the evaluation context would be the following activity-dependent movement rate:

$$\mu_r(\gamma_s, \alpha) \quad = \quad \begin{cases} 1.0 & \text{if } \alpha = \mathsf{strolling} \\ 2.0 & \text{if } \alpha = \mathsf{walking} \\ 5.0 & \text{if } \alpha = \mathsf{running} \end{cases}$$

Thus the first three functions in the evaluation context determine probabilities, weights and rates that supply quantitative information about the behaviour of actions. The fourth function μ_u performs global updates, either of the attributes in the global store or of the collective by adding new components. These updates include the usual initialisation of variables, incrementing counters, or accumulating totals.

2.1 CaSL

As is standard for process calculi and algebra, Carma is the formal and mathematical language developed for collective adaptive system modelling. However, when it comes to implementing models and simulating and analysing them, a text-based language suitable for input is required. The language accepted by the Carma Eclipse Plug-in is called CaSL (Carma Specification Language). It allows for the declaration of components and the environment as in the definition of Carma but it also gives additional features that are necessary when making a model concrete for simulation. In particular, it allows for constants and functions to be defined to support the definition of models. In addition to this, it adds a layer of typed data structures including enumerations, record types, and heterogeneous collections such as sets and lists. This provides a level of type security which is not offered by process calculi with untyped value-passing. Furthermore, the CaSL language has an explicit spatial syntax to describe space, an important feature determining the behaviour of many CAS. This allows the definition of nodes (either as coordinates or names) and links between these nodes. There is also syntax to support the use of this space, in particular, a way to refer to both the pre-set and post-set of a node, which then permits a generic definition of moving components that can traverse over any spatial structure specified. Taken together, these additional language features in CaSL provide a basis for strong static analysis of models, catching modelling errors at compile-time which would not be detected in modelling languages without this kind of support for representation of typed data and spatial structure.

CaSL provides a wrapper around the Carma process calculus adding non-essential (but useful) features such as data types and data structures, functions, and the ability to specify real-valued *measures* of interest over the model. In some modelling languages measures of interest or Markov reward structures are defined externally to the model but in Carma and languages such as CASPA [**?**], PRISM [**?**] and ProPPA [**?**], the specification of measures of interest and reward structures is included in the modelling language itself.

2.2 The Carma Eclipse Plugin

The Carma Eclipse Plugin is an integrated development environment for Carma models. It provides a helpful syntax-aware editor for CaSL, implemented in the XText editor framework [**?**]. Given a Carma model, the Carma Eclipse Plug-in compiles the model into a set of Java classes which are linked with the Carma simulator classes to provide a custom simulator for this specific model. The compiled Java code is executed to compute the measures of interest from an ensemble of simulation runs. The operational semantics of Carma are defined in FuTS style [**?**] and define the semantics of each model as a time-inhomogeneous continuous-time Markov chain (ICTMC). The behaviour of these ICTMCs is simulated using the Carma Eclipse Plug-in. The simulator uses a *kinetic Monte-Carlo* algorithm to select the next simulation event to fire and draws from the appropriate weighted random number distribution to determine the duration of the event. The simulation state is updated as specified by the event which was fired and the simulation proceeds forward until a pre-specified simulation stop time is reached. This is a standard approach and research is being made on how to improve these methods in order to speed up the simulation of complex models.

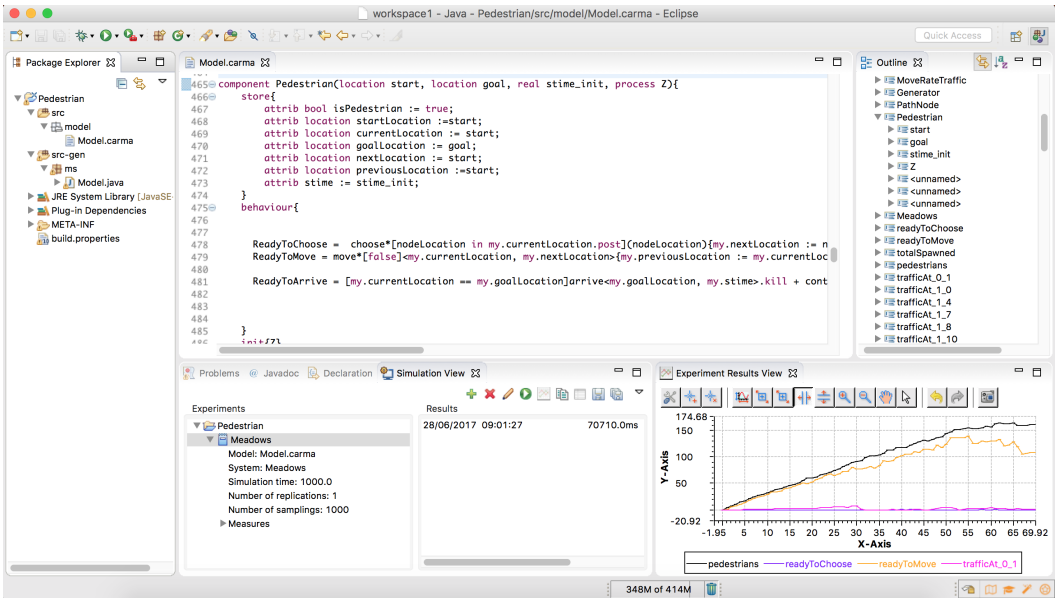


Fig. 1. A screen shot of the interface of the CARMA Eclipse Plug-in.

The measure functions defined by the modeller are passed into the simulation environment and provide a view onto the raw simulation results at intervals which are specified by the modeller. The Apache Commons Math Library is used within the Plug-in to perform statistical analysis of the data. The *Simulation Laboratory View* provided by the CARMA Eclipse Plug-in acts as an electronic laboratory notebook, recording details of the simulation studies which have been performed. Simulation experiments are composed and launched from the user interface via the CARMA Simulation View. Results are plotted directly into the Experiments Results View or saved to a file for post-processing.

A screen shot of the graphical user interface of the CARMA Eclipse Plug-in is presented in Fig. **??**.

The CARMA Eclipse Plug-in is available from the QUANTICOL SourceForge repository[1]. After installation it can be kept up-to-date using the standard mechanism in Eclipse to check for updates. As an alternative to the CARMA Eclipse Plug-in, the CARMA command-line simulator can be used to schedule a series of experiments which will then run without user interaction (for example, on a compute server). Source code for the CARMA command-line simulator is available from the QUANTICOL GitHub repository[2]. More information about the QUANTICOL project which created these tools is available from the project web site[3]. The CARMA models presented in this paper are available for download[4].

3 FORMAL LANGUAGES FOR COLLECTIVE ADAPTIVE SYSTEMS

The CARMA language provides high-level language constructs for describing communicating processes. The language has a stochastic semantics expressed in terms of continuous-time Markov chains. CARMA contains some features which are familiar from languages such as Bio-PEPA [**?**

[1] http://quanticol.sourceforge.net
[2] https://github.com/Quanticol/CARMA
[3] http://www.quanticol.eu
[4] http://blog.inf.ed.ac.uk/quanticol/pedestrian-movement/

], PRISM [?] and the Attributed π-calculus [?]. In this section we compare Carma to these established modelling languages and highlight differences in approach between them through discussion of the language features which they offer. In addition, we consider the types of model analysis which each language supports.

Each of the languages considered here has the potential to be used to model stochastic systems with mobile populations of individuals but language design decisions, the choice of language features, and underlying analysis mechanisms can make one of the languages better-suited for a particular modelling problem than the others. As examples of modelled systems, Bio-PEPA has been used to model scenarios where safe movement of people is an important factor in systems including *emergency egress* [?] and *crowd formation and movement* [?]. PRISM has been used to model *dynamic power management controllers* [?] and *human-in-the-loop UAV mission planning* [?]. Attributed π-calculus has been used to model *spatial movement in phototaxis* [?], and *cooperative protein binding in gene regulation* [?]. Carma has been used to model a number of spatial CAS including *carpooling* [?], *taxi movement* [?] and *ambulance deployment* [?].

Inter-process communication in Carma is *attribute-based*; communication partners are determined dynamically as the model evolves through state-to-state transitions. Communication in the Attributed π-calculus is similarly dynamic. In contrast, the communication partners of Bio-PEPA and PRISM components are determined statically, and do not change as state-to-state transitions occur. Additionally, Carma and the Attributed π-calculus support value-passing communication whereas the Bio-PEPA and PRISM languages do not.

The Carma language and the PRISM language are explicitly-typed. Types such as boolean, integer and real are ascribed to variables in the language by the modeller, or inferred by the language type-checker. In contrast, types in Bio-PEPA and the Attributed π calculus are implicit. Explicitly-typed languages can make the modeller's intentions more obvious, when, for example, expecting to receive an initial integer value instead of a real value.

Carma provides guarded process definitions which are used in a similar way to the guarded commands found in the PRISM language. The Attributed π-calculus does not support these directly. Bio-PEPA has no boolean expressions at all and hence no guarded expressions of any kind. Guarded process definitions allow declarative descriptions of the relationships between locations in a network and we have used this description mechanism comprehensively here.

In common with PRISM, Carma provides strong support for encapsulation, with variable declarations being local to an enclosing structure. In PRISM this structure is a *module*, whereas in Carma it is a *component*. A structuring mechanism for definitions such as this is not found in Bio-PEPA or the Attributed π-calculus where declarations of rate functions, channel names, process definitions or species definitions have global scope.

Differently from the other languages considered here, Carma treats location and space as an aspect of a model which can be described separately from the detailed model dynamics. Through the provision of a graphical editor for Carma [?], space, location, and connectivity can be treated separately from logic, communication, and synchronisation. This separation of concerns may make it easier to maintain a model of a system when the spatial structure of the system changes.

The primary analysis method for Carma models is simulation. This is also the case for Bio-PEPA and the Attributed π-calculus whereas analysis of PRISM models is typically through probabilistic model-checking. Model-checking suffers from the disadvantage that it requires a representation of the model state-space which limits the applicability of the method to small-scale systems because of the well-known *state-space explosion* problem which occurs when trying to represent all reachable states of a system of concurrent processes in an action-interleaving representation of concurrency. Analysis via simulation (as in Carma, Bio-PEPA and the Attributed π-calculus) avoids the need

to represent the full state-space explicitly but at the cost of long simulation run-times. Bio-PEPA restricts the types of simulation events and kinetic functions which are available to the modeller in order to be able to use accelerated stochastic simulation algorithms from the Gillespie family of Monte Carlo methods. CARMA and the Attributed π-calculus allow general simulation events and kinetic functions but provide distributed execution of independent simulations in order to be able to reduce the time to evaluate an ensemble of simulation runs.

CAS, and in particular crowd movement, has been considered in the aggregate programming approach of Beal and Viroli [**?**]. This is a layered approach to developing distributed applications over a cooperating collection of devices. The underlying theory of this approach is defined by the field calculus which captures a notion of distribution in space and formalises which devices can communicate. At the top level, aggregate APIs enable the actual communication between real devices. This approach has been used to provide crowd safety services whereby dangerous levels of crowding can be identified, and appropriate messages can be sent to mobile phones to give individual instructions of what to do, leading to successful dispersion [**?**]. Aggregate programming has a different aim to that of CARMA, namely developing distributed applications rather than modelling CAS scenarios.

4 MODELLING OF PEDESTRIAN MOVEMENT

How pedestrians move through space and along walkways has been an area of research interest for many years, and different approaches to modelling these phenomena have been taken, often based on concepts developed in physics [**? ? ?**]. There are two main areas of interest overall: normal conditions and emergency or panic conditions. The current paper focusses on normal conditions rather than the more extreme and unusual case.

Modelling approaches can be classified as *microscopic*, *mesoscopic* and *macroscopic*. The two extremes are moderately straightforward to define, but the middle one is more complex. In the case of microscopic models, each pedestrian is modelled individually, and is typically located in two-dimensional space. This can be done through agent-based models, cellular automata, magnetic force models or social force models [**? ? ?**]. For cellular automata, discrete two-dimensional space (a grid or lattice) is used, whereas the other approaches consider continuous space.

In contrast, macroscopic models consider densities of pedestrians rather than individuals at specific locations in space. They are typically defined by partial differential equations, allowing for the continuous variation of time as well as location in space.

The third category, mesoscopic is used in a number of different ways. For example, Bellomo and Bellouquid define a multi-scale model, where micro-scale interactions are described [**?**]. These lead to a mesoscopic kinetic model from which macroscopic equations can be obtained by considering asymptotic limits. This means that the macroscopic description emerges from the microscopic interactions rather than being defined *a priori*. An alternative use is where physical and logical groups of pedestrians are used within the simulation [**?**].

Our use of the term mesoscopic is influenced by modelling of molecular phenomena, in particular, diffusion of molecules in three-dimensional space. In this domain, microscopic and macroscopic are used as above. In the former, individual molecules are modelled in three-dimensional space, including their collisions; and in the latter, PDEs are used to describe changes in density of the molecular species involved. An alternative approach is to assume well-mixedness and use stochastic simulation or ODEs based on the Chemical Master Equation (CME) to describe the changes in quantities of chemical species but this does not take space into account. The mesoscopic approach involves dividing three-dimensional space into areas of volume known as voxels, and the CME (or suitable approximations) assume that within each voxel, species are well-mixed and then can

be used to express this interaction. However, this approach takes into account the fact that some molecules will move between voxels, and determines this based on a Reaction-Diffusion Master Equation (RDME) [**?** **?**].

We take a similar approach, although without defining a master equation. We consider pedestrians moving along paths but we do not model the individual movement of each pedestrian as they are moving along the path, only as they move from one path segment to another. Their movement within a path segment is determined by an exponentially-distributed duration and this duration is state-dependent, is the sense that it is dependent on the number of pedestrians on that path segment.

5 BASIC PEDESTRIAN MODEL

Our Carma model of basic pedestrian movement is presented in Figures **??** and **??**. It assumes that there are two types of pedestrian, A and B, and that P and Q are variables of type pedestrian. The two *Generator* components generate pedestrians at two different locations (on opposite sides of the graph), and the pedestrians move from their origin side to the opposite side. The pedestrians' behaviour is split into two alternating processes. When at a node pedestrians are in the state *Choose*. After the next edge has been chosen, pedestrians go to the state *Move*. Once a pedestrian has reached its goal, the count for that type of pedestrian is incremented ($count_P$ in Figure **??**) and the time taken for traversal is added to the total time so that the average traversal time can be calculated for each pedestrian type. These variables are global and form part of the environment.

- **ExistsPath**(P, ℓ, k) is a Boolean function that determines if an edge exists between a pedestrian's current position ℓ and another node k, hence a $\text{move}^\star_k$ action can only occur when such an edge exists.
- **AtGoal**(P, ℓ) is a Boolean function that checks if the pedestrian has reached its goal, hence $\text{fin}^\star$ can only occur once the destination has been reached. After this the pedestrian does not move any more.
- **ArrivalRate**(P) is a function that returns λ_P, the arrival rate for pedestrians of type P. These rates are defined as constants in Figure **??**.
- **Start**(P) defines the initial location of a new pedestrian depending on its type.

A function that is not directly related to the graph structure is **MoveRate**$(P, \ell, k, A_{\ell,k}, B_{\ell,k})$ which determines the rate of movement along a particular edge (ℓ, k), and can take additional parameters that can affect this rate such as the current count of other pedestrians of the same or different type. We use the following definition that uses the numbers of pedestrians of the other type on the current edge to reduce the movement rate.

$$\textbf{MoveRate}(P, \ell, k, A_{\ell,k}, B_{\ell,k}) = \begin{cases} move_A/(B_{\ell,k}+1) & \text{if } P = A \\ move_B/(A_{\ell,k}+1) & \text{if } P = B \end{cases}$$

where $A_{\ell,k}$ is the number of A pedestrians on the edge and $B_{\ell,k}$ is the number of B pedestrians on the edge, and $move_P$ is a basic movement rate for each pedestrian type.

Another function is **ChooseRate**$(P, \ell, k, A_{\ell,k}, B_{\ell,k})$ which determines the rate (in effect, the probability) of choosing a particular edge (ℓ, k) for the next move. Similar to the definition of **MoveRate**, our definition takes into account the number of pedestrians of the other type on that edge, such that pedestrians will favour edges with lower traffic.

$$\textbf{ChooseRate}(P, \ell, k, A_{\ell,k}, B_{\ell,k}) = \begin{cases} fast/(B_{\ell,k}+1) & \text{if } P = A \\ fast/(A_{\ell,k}+1) & \text{if } P = B \end{cases}$$

Store of *Pedestrian* **component:**

P	pedestrian type
ℓ	current location
$n\ell$	next location
stime	time of arrival

Behaviour of *Pedestrian* **component:**

$$Choose \stackrel{\text{def}}{=} \sum_{k \in V} \left[\mathbf{ExistsPath}(P, \ell, k)\right] \left(\mathsf{choosePath}^{\star}_{k}[\bot]\langle\rangle\{\mathsf{my}.n\ell \leftarrow k\}.Move\right) + \left[\mathbf{AtGoal}(P, \ell)\right] \left(\mathsf{fin}^{\star}[\bot]\langle\rangle.\mathbf{nil}\right)$$

$$Move \stackrel{\text{def}}{=} \left(\mathsf{move}^{\star}[\bot]\langle\rangle\{\mathsf{my}.\ell \leftarrow \mathsf{my}.n\ell\}.Choose\right)$$

Initial state of *Pedestrian* **component:** *Choose*

Store of *Generator* **component:**

P	pedestrian type

Behaviour of *Generator* **component:**

$$Arr \stackrel{\text{def}}{=} \mathsf{arrive}^{\star}[\bot]\langle\rangle.Arr$$

Initial state of *Generator* **component:** *Arr*

Fig. 2. The *Pedestrian* and *Generator* components of the basic model

where $A_{\ell,k}$ is the number of A pedestrians on the edge and $B_{\ell,k}$ is the number of B pedestrians on the edge. It is assumed that decisions are made relatively quickly, thus we use a constant factor $fast \geq 100$ to make $\mathsf{choosePath}^{\star}_{k}$ a fast action. We have chosen to work with these functions as they meet our expectations of the behaviour of pedestrians in these circumstances. We do not have to access data to take a different approach.

Figure **??** specifies the four functions $(\mu_p, \mu_w, \mu_r, \mu_u)$ known as the evaluation context. Probabilities and weights on activities are not used in this model so the μ_p and μ_w functions are trivially constant functions.

As mentioned above, the model is mesoscopic. Each edge in the graph represents a path segment, and the path segment that each pedestrian is on is determined by their current and next position The model only records that they are on a specific path segment and there is no explicit knowledge about where on the path segment a pedestrian is, only a time duration for traversing the segment. This is an abstraction that allows for efficient modelling as it is not necessary to know the specific location when traversing a path segment.

5.1 Model instances

Four instances of this Carma model are shown in Figure **??**. These show instantiations of the general Carma model from the previous section with increasing size and shape complexity and the same rates for traversing an edge. The central repeating features of the path network are the cross-bars in the centre of the network. In the simplest instance we have only one cross-bar and

Constants:

V	set of coordinate pairs representing nodes in the graph
λ_P	arrival rate for pedestrians of type P
$move_P$	movement rate for pedestrians of type P

Measures:

$average_P$	average time for traversal by pedestrians of type P

Global store:

$count_P$	number of P pedestrians that have completed the traversal
$total_P$	total time for all completed P pedestrian traversals

Evaluation context:

$$\mu_p(\gamma_s, \gamma_r, \alpha) = 1$$
$$\mu_w(\gamma_s, \gamma_r, \alpha) = 1$$

$$\mu_r(\gamma_s, \alpha) = \begin{cases} \mathbf{ArrivalRate}\big(\gamma_s(P)\big) & \text{if } \alpha = \mathsf{arrive}^\star \\ \mathbf{MoveRate}\big(\gamma_s(P), \gamma_s(\ell), k, A_{\ell,k}, B_{\ell,k}\big) & \text{if } \alpha = \mathsf{move}_k^\star \\ \mathbf{ChooseRate}\big(\gamma_s(P), \gamma_s(\ell), k, A_{\ell,k}, B_{\ell,k}\big) & \text{if } \alpha = \mathsf{choosePath}_k^\star \\ \lambda_{fast} & \text{otherwise} \end{cases}$$

$$\mu_u(\gamma_s, \alpha) = \begin{cases} \big\{\{\}, \big(Pedestrian, \{P \leftarrow \gamma_s(P), \ell \leftarrow \mathbf{Start}(\gamma_s(P)),\ stime \leftarrow \mathsf{now}\}\big)\big\} & \text{if } \alpha = \mathsf{arrive}^\star \\ \big\{\{count_{\gamma_s(P)} \leftarrow count_{\gamma_s(P)} + 1, total_{\gamma_s(P)} \leftarrow total_{\gamma_s(P)} + (\mathsf{now} - \gamma_s(stime))\}, 0\big\} & \text{if } \alpha = \mathsf{fin}^\star \\ \{\{\}, 0\} & \text{otherwise} \end{cases}$$

Collective:

$$PedAB \stackrel{\text{def}}{=} \big(Generator, \{P \mapsto A\}\big) \parallel \big(Generator, \{P \mapsto B\}\big)$$

Fig. 3. Environment and collective of the basic model

we describe this instance as having height 1 and width 1, representing it as instance 1×1. The height of an instance is the number of cross-bar elements from top to bottom, and the width is the number of cross-bar elements from left to right. As the cross-bar structure is repeated we have instances 1×2 (which has height 1 and width 2), 2×1 (which has height 2 and width 1), and 2×2, depending where the additional structure is added into the network.

An increase in the width of the network has the obvious consequence that journeys across the network take longer. An increase in the height of the network has the consequence that pedestrians are offered an increased choice of routes, with the implicit consequence that individual paths are less congested (because there are more of them on offer). The edges are assumed to be equally long and thus the time to traverse them is the same under comparable conditions.

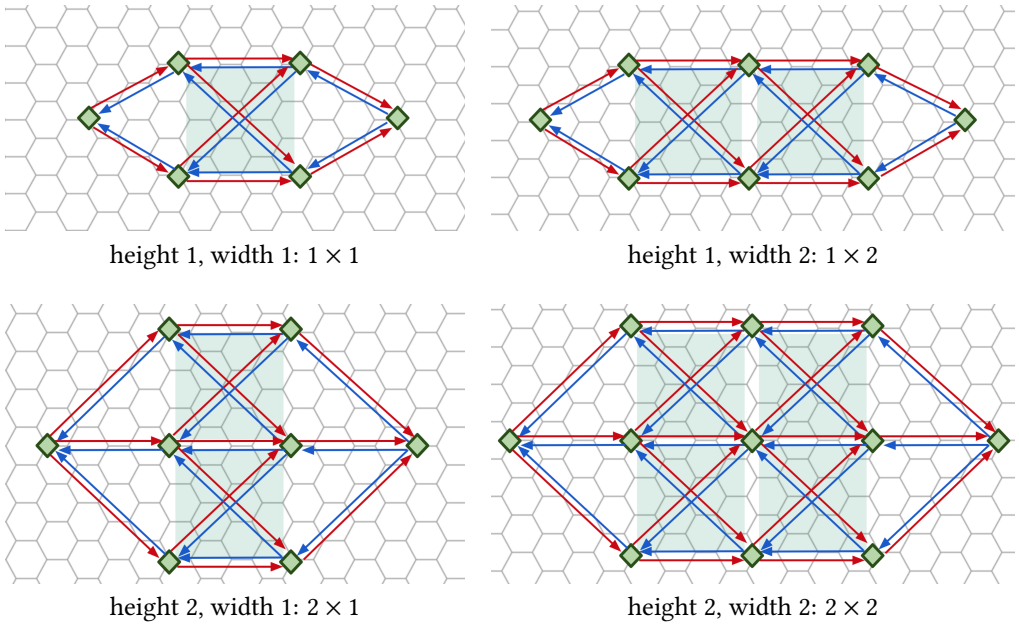


Fig. 4. Four model instances of increasing size and complexity: height indicates the number of crossbar elements from top to bottom and width the number of crossbar elements from left to right.

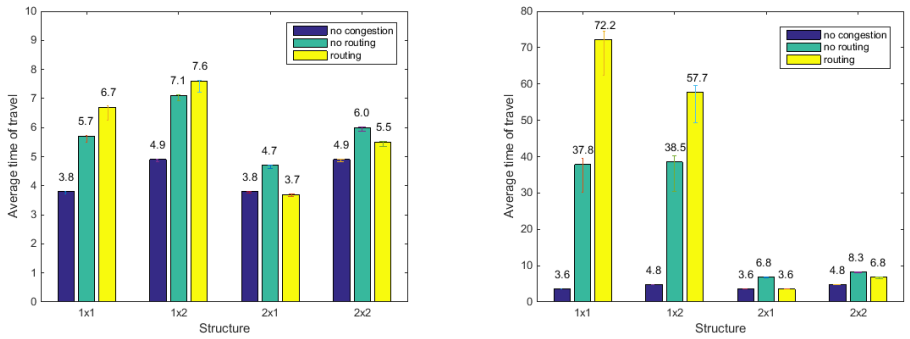
In each instance of the network of paths there are two sub-networks which constrain the movement of the pedestrians of type A and type B. Pedestrians of type A are restricted to the red sub-network and must cross the network from left to right. Pedestrians of type B are restricted to the blue sub-network and must cross the network from right to left. The networks illustrated in Figure **??** are symmetric but this is of no particular significance and it would pose no difficulty to work with networks which were not symmetric, which we will show with the Meadows example later in this paper.

5.2 Design of experiments

We designed a suite of experiments to explore the behaviour of the model. To provide a baseline for average travel time we investigated the travel time in the presence of only one type of pedestrian (thereby giving a model which has no congestion). Thereafter we investigated the models with congestion in the presence or absence of pedestrian routing. When routing is present, only one starting route has a non-zero rate and the non-zero rate is assigned in order to direct pedestrians away from each other.

The two main parameters which influence the behaviour of the model are the movement rates and arrival rates. Since the movement rates depend on congestion, it was important to set plausible base rates for the non-congested case. Therefore, parameters were calibrated under the assumption that all links are equidistant (approximately 100m long) and using the results of a study conducted by [**?**], which suggests an expected pedestrian speed of $1.34m/s$ under normal conditions.

Having set base rates for pedestrian movement, then the focus of the experiments was to investigate traffic under different arrival rates, i.e. different numbers of pedestrians in the network. In a first experiment arrival rates are low, which should lead pedestrians to pass through the



(a) Low traffic (b) Higher traffic

Fig. 5. Average travel time results from the experiments on structure and network usage

network smoothly. In a second experiment arrival rates are doubled, which we expect to cause more congestion in the network and thus longer average travel times.

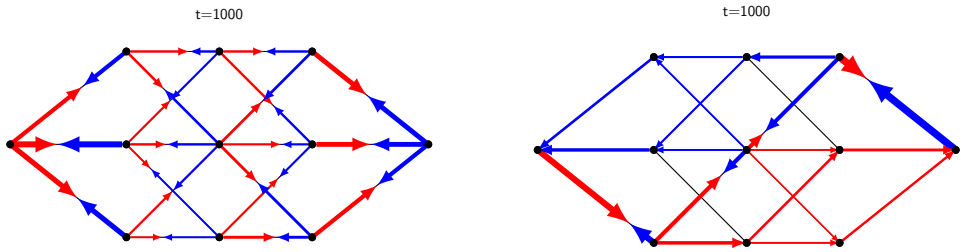
So far, arrival rates of both pedestrian groups had been equal. Therefore, we conducted a third experiment, in which we explored how the arrival rate of pedestrian type A affects the average travel time of B. In order to do that, the arrival rate of A was set to $s * arrivalRate_B$, $s \in \{1, 2, 3, 4, 5\}$.

5.3 Analysis

We are able to obtain values for the length of time it takes each pedestrian to traverse the whole system, even though our mesoscopic approach abstracts away from the details of where each pedestrian is on a path segment. We can then compare on average how long it takes different types of pedestrian to move from their entrance point to their exit point for different scenarios.

The results from our first two experiments are presented in Figure **??**. For each experiment we have three results (no congestion, routing, and no routing) for each of the four model instances considered (1×1, 1×2, 2×1, and 2×2). Figure **??** shows the results for low overall traffic. An inspection of the results shows that, unsurprisingly, for any structure the best average travel times are obtained when there is no congestion in the network. As anticipated, networks with greater height have shorter average travel times because they have greater capacity, due to the inclusion of additional routes (thus 2×1 results are better than 1×1 results, and 2×2 results are better than 1×2 results). In these networks routing is advantageous, whereas when applying routing to narrow (low height) networks, the opposite behaviour is observed, in fact routing leads to an increase in the average travel time. Similar results were obtained for the case of high traffic, which are displayed in Figure **??**. Here, where average travel times in narrow networks are already extremely high in the absence of routing, they increase even more when routing is present. The observation that routing increases travel time is contradictory to our expectations and this is discussed further below.

Congestion in the 2×2 network is illustrated in Figures **??** and **??**. Red and blue arrows indicate direction of movement of the two pedestrian types. The overall number of pedestrians on a particular edge is represented by line width, while the proportion of A and B on an edge is represented by arrow length. When there is no routing present, congestion especially occurs near start and end nodes where all paths connect. All other edges (the ones closer to the center of the network) are less congested, since traffic is equally split over all possible paths. In comparison to that, when routing is enabled, pedestrian groups are guided away from each other, such that most edges are



(a) Without routing (b) With routing

Fig. 6. Congestion on network edges at time $t = 1000$ for the 2×2 topology

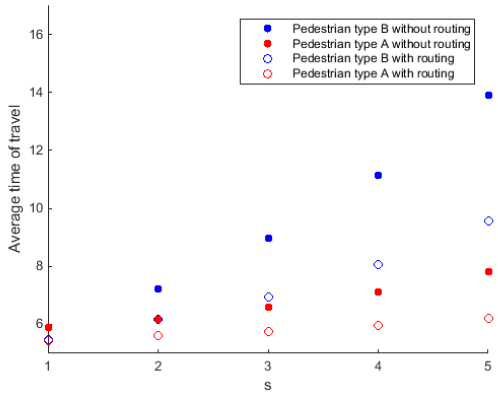


Fig. 7. Average travel time of B against arrival rate of A

only used by one pedestrian group. Congestion occurs only near start and end points and on the main diagonal, as not all individuals choose the path with lowest traffic.

The results of our third experiment on the 2×2 network are shown in Figure **??**. It can be seen that an increase in the amount of pedestrian type A leads to a significant slowdown of pedestrian type B, whereas the average travel time of type A rises only slightly. Using pedestrian routing helps to reduce average travel time in all cases.

5.4 Discussion

As mentioned above, some of the results we obtained were counter-intuitive, and differed from our expectations. In fact, it turns out that our expectations were incorrect as we had not fully comprehended a particular aspect of our model. The functions that determine which path to choose and how fast movement is possible along a path are dependent on local information. In the case of movement speed, it is reasonable just to use the quantity of other pedestrians to determine this. However, for the choice of path, using only the local information of the paths that can be chosen

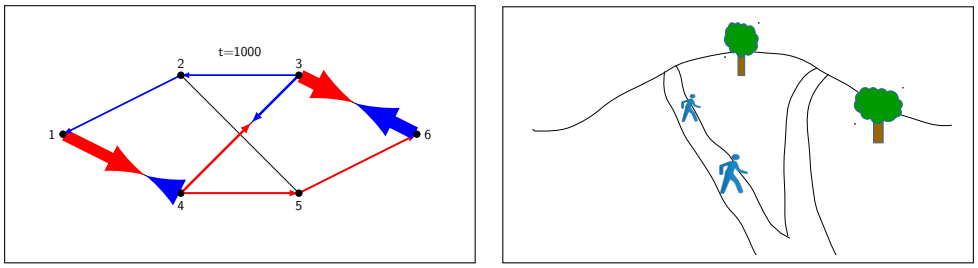


Fig. 8. Congestion of edges for 1×1 network (left) and illustration of path visibility and how it may affect choice (right)

imposes a notion of visibility onto the model, which can also be used to infer the topography of the landscape over which the paths are defined.

Consider the 1×1 model as given in Figure **??**, a pedestrian entering from the right (node 6) in the routing case, will be directed along the upper branch (to node 3) and one entering from the left (node 1) will be directed along the upper branch (to node 4). This pedestrian at node 4 will then have a choice between the upper (to node 3) or lower branch (to node 5), both of which will appear reasonable because there will be little oncoming traffic on either. In fact, on the lower path there will be none, because of the routing applied to pedestrians to the right, but other will also be low, so some proportion will take the upper path to node 3 and then take a very long time to reach node 6 as they will be facing the full flow of *all* pedestrians coming in other direction, and their average time to traverse the last link increases, as does the average time for the pedestrians that are traversing their first link. Hence routing causes some pedestrians to take a path that looks good but is not (node 4 to 3). This does not occur to the same extent in the $2 \times n$ cases because even though a similar poor choice can be made, it will have a less pronounced effect.

The reason why this occurs is that looking at only the next paths ahead does not give complete information. This arises in practice when pedestrians have limited visibility on the paths ahead. To obtain better flow, there are a number of solutions including signposting at the point where the poor choice is made, and allowing path look-ahead of more than one path segment so that congestion further ahead can be seen. Another way to consider this is to view the paths as being laid over a landscape that is hilly – the poor choice is made because the hilly landscape prevents a pedestrian from seeing further ahead as illustrated in Figure **??**.

These results differ from those in our previous paper [**?**] where we did not observe this counter-intuitive behaviour. The explanation for this is the fact that our model now uses path congestion to determine behaviour rather than node congestion as in the prior work. The change was made to reflect human behaviour better; however, it also gave us an improved understanding of model behaviour, and a more general model could then be developed which captures explicit notions of visibility.

The model and our results demonstrate that mesoscopic modelling is possible for this pedestrian movement scenario. Moreover, by abstracting from the specific location a pedestrian is on a path segment, we ensure that the simulation of the model is feasible, both in terms of model size and time required for simulation. It is beyond the scope of this paper to compare our results with those of a microscopic or macroscopic model.

In our second model which considers the Meadows in Edinburgh, an additional piece of information is used to determine which path to choose and that is the distance to the goal node (which varies between pedestrians unlike in the basic model).

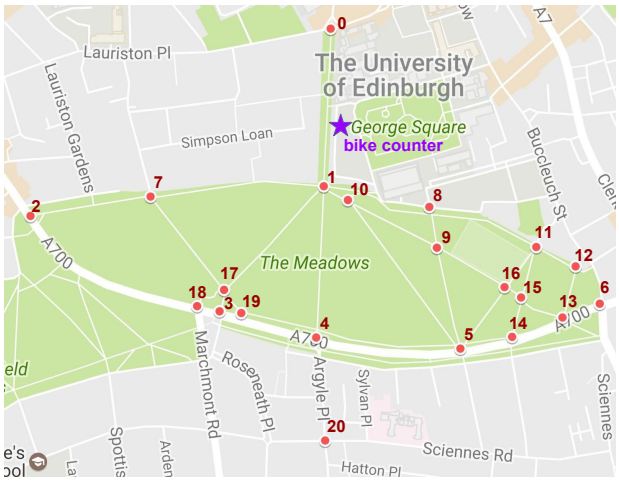


Fig. 9. The Meadows public park in Edinburgh, Scotland. The locations of the intersections used in the system are marked in red.

6 CASE STUDY: THE MEADOWS

The second model we present provides a more sophisticated model of an existing network of pedestrian paths and cyclepaths in the Meadows public park in Edinburgh, Scotland.

The aim of modelling this scenario was to explore the extent of expressiveness of the CARMA language, rather than to compare the model's predictions to data. This is because data suitable for such a comparison are not available.

Unlike the basic model, we based this case study on real-world geographic data. We used the Google Maps API to obtain the location of 20 path lane intersections in The Meadows area (see

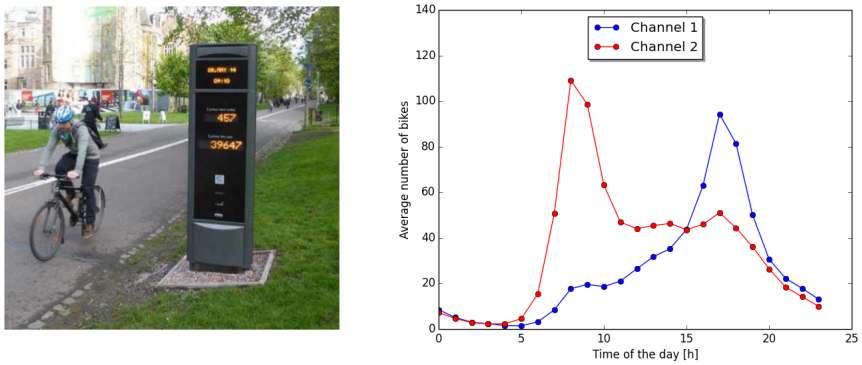


Fig. 10. The data available from the City of Edinburgh council's website (http://www.edinburghopendata.info/vi/dataset/bike-counter-data-set-cluster/resource/ffbeb785-a19f-4788-bc27-892b024a6750) shows which times of the day have greater traffic of bikes. The bike counter, located near the north segment of the Middle Meadow Walk (see Fig. ??) is able to detect and count bikes moving in opposite directions separately (Channel 1 and Channel 2). The data shows increasing traffic of bikes in the morning and afternoon.

Fig. ??) and used these to generate the structure of connections and nodes in the model. The Google Maps API provides the latitude and longitude for each selected point in accordance with the World Geodic System 1984 (WGS 84). These values were then converted using the Universal Transverse Mercator (UTM) projection in order to obtain the x and y cartesian coordinates on the plane, in the unit of metres.

6.1 The Carma model

In the basic model presented in Section ?? the start and finish (goal) location of each pedestrian is predetermined by the graph representing the network of the paths. In the more complex example of the Meadows, multiple start and finish locations are allowed and all connections are bidirectional. For this reason, in order to mimic the behaviour of the real system, in which commuters cross the park with the intention to get from one place to another, each Pedestrian is assigned a start and finish location when they first appear in the system. The probability of choosing a given location as a start or a finish (goal) location for a given component depends on the time of the day to reflect the data published by the City of Edinburgh, obtained from the bike counter (Fig. ??). We used a sum of Gaussian functions to represent these traffic changes throughout the day, which is a reasonable fit to the available data.

The definitions of the components are shown in Fig. ?? and ??. The definition of the environment, evaluation context, and the collective of the Carma model are shown in Fig. ??. In particular, the evolution rule involves a number of functions that are described in the next subsections and shown in Fig. ??, ??, ??, ??.

6.1.1 Spawning new Pedestrians.

Spawn rate. The rate at which new Pedestrians arrive into the system depends on the current time of the day, to reflect the changing intensity of traffic. The data from the city bike counter shows a trend of increasing bike traffic in the rush hours (Fig. ??). The average bike traffic increases in one direction in the morning hours, and in the opposite direction in the late afternoon, this presumably being the result of people travelling to and from the city centre for work. It also shows a small increase around midday (lunchtime). In our model we used a traffic intensity pattern loosely based on these observations. The function we used is a sum of three Gaussian functions having the standard deviation of 1 hour and the means at 9:00 am, 12:30 pm, and 5:00 pm, respectively. The reason for using this simplified function of traffic intensity for the first version of the model, over one that fit more accurately to the data, is that it made the influence of the traffic intensity on the results of the simulation more visible and easy to control. The spawn rate formula is shown in Fig. ??.

$$\mathbf{Rate}_{\mathbf{spawn}}(t) = (\mathcal{N}(t, \mu_1, \sigma^2) + \mathcal{N}(t, \mu_2, \sigma^2) + \mathcal{N}(t, \mu_3, \sigma^2) + 0.1) \cdot C,$$

$$\text{where: } \mathcal{N}(t, \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi\sigma^2}} e^{\frac{(t-\mu)}{2\sigma^2}},$$
$$\sigma = 60\text{min}, \mu_1 = 540\text{min}, \mu_2 = 750\text{min}, \mu_3 = 1020\text{min}, C = 400.0.$$

Fig. 11. The rate at which new components arrive in the system depends on the hour of the day. The three values of the mean of the gaussian function μ_1,μ_2, μ_3, represent three time points of the day: 9 am, 12.30 am and 5 pm, respectively, in the unit of minutes. σ is the standard deviation of the Gaussian function with mean μ_i. C is a dimensionless scaling factor, introduced in order to obtain a realistic spawn rate value.

Spawn locations. Examination of the data from the city bike counter suggests that new bikes arriving into the system should not be uniformly distributed at all possible entry points, at all times of the day. In the morning rush hours, cyclists are much more likely to be travelling in one direction and in the afternoon rush hour, the opposite. The bike counter, from which this data has been obtained, is located in the north part of The Middle Meadow Walk, the widest path through the Meadows, connecting the south-most and the north-most parts of the park (see Fig. **??**). There is no available data about the traffic changes on all of the other paths in the park, however based on everyday observations, we have assumed the two extreme points of The Middle Meadow Walk to be the most likely entry points to the system during the rush hours. In our model, every time a new Pedestrian component is introduced to the system, it is assigned a start and a goal location. New Pedestrian components are instantiated as a result of the Generator component performing the spawn* action, which uses the start and goal location values saved in the store of the Generator component. At each location there is a PathNode component, which attempts to perform two actions assignGoal* and assignStart*, in order to communicate with the Generator component and update the values of the start and goal location in its store with the PathNode's location. The probability of receiving this message (μ_p, see section **??**) can vary for each location, and the resulting value of start or goal location is determined by the the PathNode component that manages to communicate with the Generator component first. As a result, multiple PathNodes compete with one another in order to decide which one will become the start or the goal location of the new Pedestrian instance. The probability of performing a particular broadcast output action in Carma is computed using a propensity function associated with each communication candidate. The propensity function can be parametrized using values from the store of the sender and the receiver as well as on the global store of the system. In the presented model we used the propensity functions shown in Fig. **??**. All of the possible entry locations $n_{loc} \in [0, 2, 3, 5, 6, 7, 8, 20]$ (see Fig. **??**) have the propensity value ≥ 1.0. Other locations have the propensity value of 0.0, which means that they are never chosen as the start or goal location. Two particular locations, loc_0 and loc_{20}, situated at the extreme points of The Middle Meadow Walk have a propensity that varies over the time of the day, as shown in Fig. **??**. In this way, when spawning a Pedestrian, the propensity of

$$\mathbf{Pr}_{goal}(n_{loc}, t) = \begin{cases} \mathcal{N}(t, \mu_1,\, \sigma^2) \cdot C + 1.0 & \text{if } n_{loc} = 0, \\ \mathcal{N}(t, \mu_3,\, \sigma^2) \cdot C + 1.0 & \text{if } n_{loc} = 20, \\ 1.0 & \text{if } n_{loc} \in [2, 3, 5, 6, 7, 8], \\ 0.0 & \text{otherwise.} \end{cases}$$

$$\mathbf{Pr}_{start}(n_{loc}, t) = \begin{cases} \mathcal{N}(t, \mu_1,\, \sigma^2) \cdot C + 1.0 & \text{if } n_{loc} = 20, \\ \mathcal{N}(t, \mu_3,\, \sigma^2) \cdot C + 1.0 & \text{if } n_{loc} = 0, \\ 1.0 & \text{if } n_{loc} \in [2, 3, 5, 6, 7, 8], \\ 0.0 & \text{otherwise.} \end{cases}$$

$$\text{where: } \mathcal{N}(t, \mu,\, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi\sigma^2}} e^{\frac{(t-\mu)}{2\sigma^2}},$$
$$\sigma = 60\text{min},\ \mu_1 = 540\text{min},\ \mu_3 = 1020\text{min},\ C = 10.0.$$

Fig. 12. The two values of the mean of the Gaussian function μ_1, μ_3, represent two time points of the day: 9 am and 5 pm, respectively, in the unit of minutes. σ is the standard deviation of the Gaussian function. C is a dimensionless scaling factor, introduced in order to obtain a realistic propensity value.

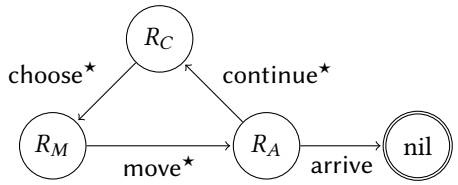


Fig. 13. The states of the process of the Pedestrian component. R_C is ReadyToChoose, R_M is ReadyToMove, R_A is ReadyToArrive, and reaching the nil state is equivalent to the component being removed from the system.

$$\mathbf{Rate}_{\mathbf{move}}(n_{from}, n_{to}, \mathrm{Trfc}(n_{to}, n_{from})) = \begin{cases} \dfrac{1}{Dstn(n_{from}, n_{to})} & \text{if } \mathrm{Trfc}(n_{to}, n_{from}) = 0 \\ \dfrac{1}{Dstn(n_{from}, n_{to}) \cdot \mathrm{Trfc}(n_{to}, n_{from})} & \text{otherwise.} \end{cases}$$

where:
$\mathrm{Trfc}(n_{to}, n_{from})$ is the number of pedestrians travelling from n_{to} to n_{from}
(These are travelling in the opposite direction to the pedestrian evaluating this function),
$Dstn(n_{from}, n_{to})$ is the distance between n_{to} and n_{from}.

Fig. 14. The rate at which components move from current to the next node depends on the traffic on the edge between the current and the next node, in the opposite direction. This model only reflects the case where only the people travelling in the opposite direction slow the pedestrian down.

assigning location loc_0 (most-north) as the start location reaches its peak at 5:00 pm and as the goal location - at 9:00 am. For location loc_{20} (most-south) this trend is reversed, i.e., the propensity of it being assigned as the start location of a new Pedestrian is highest at 9:00 am and as the goal location - at 5:00 pm. We used the Gaussian function (see Fig. **??**) to model this increase in propensity.

6.1.2 Movement and routing. A Pedestrian component is always trying to reach its goal location (see Fig. **??**). In order to do that, it performs a sequence of actions, which changes its state in the cycle shown in Fig. **??**. The behaviour of a Pedestrian component in the Meadows model is very similar to that of the Pedestrian component in the basic model (see Fig. **??**). A path is chosen after which the move takes place. There is also a check to see if the final destination has been reached. When the Pedestrian component is in the ReadyToChoose state, it must perform the choose* action, which will determine the next node which the component will change its current location to.

In a fashion similar to the way the start and goal locations are chosen before a new Pedestrian is created, multiple PathNode components compete with each other to decide which one of them is going to synchronize with the Pedestrian component on the choose* action, determining the next location for the Pedestrian to move to.

The propensity of synchronising with a particular PathNode on the choose* action is given by the function shown in Fig. **??**. The propensity is calculated for each potential next node. This formula was chosen in order to model intention of the Pedestrian component to move towards its goal node, while at the same time avoiding routes with heavy congestion. Let us first consider the case when the value of traffic is 0 (there are not any other Pedestrian components travelling along the same connection). In that case, the propensity depends only on the value of the $Dstn$ factor. $Dstn_d$ is a measure of how much closer the Pedestrian will get to its goal node, by moving

$$\mathbf{Pr}_{\mathbf{choose}}(n_{next}, n_{curr}, n_{goal}) = \begin{cases} \dfrac{Dstn_d}{\mathrm{Trfc}(n_{next}, n_{curr})} & \text{if } \mathrm{Trfc}(n_{next}, n_{curr}) > 0 \\ & \text{and } Dstn_d \geq 0, \\ \dfrac{-1}{Dstn_d \cdot \mathrm{Trfc}(n_{next}, n_{curr})} & \text{if } \mathrm{Trfc}(n_{next}, n_{curr}) > 0 \\ & \text{and } Dstn_d < 0, \\ \dfrac{-1}{Dstn_d} & \text{if } \mathrm{Trfc}(n_{next}, n_{curr}) = 0 \\ & \text{and } Dstn_d < 0, \\ Dstn_d & \text{if } \mathrm{Trfc}(n_{next}, n_{curr}) = 0 \\ & \text{and } Dstn_d \geq 0. \end{cases}$$

where:
n_{curr},n_{goal},n_{next} are the current, goal and (prospective) new location of a Pedestrian component,
$Dstn_d = Dstn(n_{curr}, n_{goal}) - Dstn(n_{next}, n_{goal})$,
$Dstn(n_1, n_2)$ is the distance between two given nodes,
$\mathrm{Trfc}(n_{next}, n_{curr})$ is the number of pedestrians whose current location is n_{next} and next location is n_{curr}.

Fig. 15. The propensity of choosing a particular node as the next location.

to the next node. It is computed by calculating the difference between the Pedestrian's distance to its goal from the current node and from the next potential node. If the value of $Dstn_d$ is positive, the current node is further from the goal node than the next potential node - and so, moving to the next node is desired since it brings the Pedestrian closer to its goal. The propensity value of such a move is proportional to the potential decrease in distance from the goal, represented by $Dstn_d$. If the value of $Dstn_d$ is negative, moving to the next potential node takes the Pedestrian further away from its goal node, and so the propensity value associated with this move is inversely proportional to the value of $Dstn_d$.

The same relationship between the value of $Dstn_d$ and of propensity is also used in the case when there is a non-zero traffic on the connection to the potential node. The propensity of choosing a node is always inversely proportional to the measured traffic.

The graph structure of the network of paths is incorporated into the system using a different approach than in the basic model. Rather than invoking the **ExistPath** function in order to determine whether two nodes are connected by an edge or not, a special expression from the space syntax available in the newest version of CASL is used. The loc_A *in* loc_B.*post* expression evaluates to a boolean value which is **true** if the loc_A location is in the post set of the loc_B location, and **false** otherwise.

The predicate of the choose* action uses the above syntax to ensure that there exists an edge from the Pedestrian component to the PathNode component which the Pedestrian is attempting to synchronize with.

If there is more than one PathNode in the postset of the Pedestrian component location, a given PathNode is more likely to be chosen over another if it is closer to the goal node of the Pedestrian,

Store of *Pedestrian* **component:**

startL	start location
goalL	goal location
currL	current location
nextL	next location
prevL	previous location
stime	time of arrival

Behaviour of *Pedestrian* **component:**

$$
\begin{array}{rcl}
\mathit{Choose} & \stackrel{\text{def}}{=} & \mathsf{choose}^{\star}[\ell \text{ in } \mathsf{my}.\mathit{currL}.\mathsf{post}](\ell)\{\mathsf{my}.\mathit{nextL} \leftarrow \ell\}.\mathit{Move} \\
\mathit{Move} & \stackrel{\text{def}}{=} & \mathsf{move}^{\star}[\bot]\langle \mathsf{my}.\mathit{currL}, \mathsf{my}.\mathit{nextL}\rangle\{\mathsf{my}.\mathit{prevL} \leftarrow \mathsf{my}.\mathit{currL}, \mathsf{my}.\mathit{currL} \leftarrow \mathsf{my}.\mathit{nextL}\}.\mathit{Arrive} \\
\mathit{Arrive} & \stackrel{\text{def}}{=} & [\mathsf{my}.\mathit{currL} = \mathsf{my}.\mathit{goalL}]\mathsf{arrive}[\top]\langle \mathsf{my}.\mathit{goalL}, \mathsf{my}.\mathit{stime}\rangle.\mathbf{kill} + \\
 & & \mathit{continue}^{\star}[\bot]\langle\rangle.\mathit{Choose}
\end{array}
$$

Initial state of *Pedestrian* **component:** *Arrive*

Store of *Generator* **component:**

startL	start location
goalL	goal location

Behaviour of *Generator* **component:**

$$
\begin{array}{rcl}
\mathit{AssignStart} & \stackrel{\text{def}}{=} & \mathsf{assignStart}^{\star}[\top](\mathit{start})\{\mathsf{my}.\mathit{startL} \leftarrow \mathit{start}\}.\mathit{AssignGoal} \\
\mathit{AssignGoal} & \stackrel{\text{def}}{=} & \mathsf{assignGoal}^{\star}[\top](\mathit{goal})\{\mathsf{my}.\mathit{goalL} \leftarrow \mathit{goal}\}.\mathit{Spawn} \\
\mathit{Spawn} & \stackrel{\text{def}}{=} & \mathsf{spawn}^{\star}[\bot]\langle\rangle.\mathit{AssignStart}
\end{array}
$$

Initial state of *Generator* **component:** *AssignStart*

Fig. 16. The *Pedestrian* and *Generator* components of the Meadows model

and less likely to be chosen if the edge between the current location of the Pedestrian leading to that node has more traffic than the edge leading to the other node.

This introduces a form of routing - Pedestrians are more likely to choose paths with less traffic.

The distances between nodes are pre-computed and are hard-coded into the model in the form of a globally accessible two-dimensional look-up table. When the Pedestrian component arrives in its goal node, it is removed from the system.

Moving from one node to another is a CARMA action. The rate of the action was chosen to reflect the delay caused by the distance between two nodes, as well as the congestion on the path leading from one node to another (see Fig.**??**). The value of the distance as well as the value of the measured traffic are inversely proportional to the resulting rate of the movement action. This means that, for example, on average a Pedestrian will traverse two 50 m long paths in the same time as it would traverse a single 100 m long path.

Store of *PathNode* component:

nodeL	location of node
arrived	number of pedestrians that have arrived at this node as a goal
timeSum	sum of times taken by pedestrians arriving at this node as a goal

Behaviour of *PathNode* component:

$$Advert \stackrel{\text{def}}{=} \mathsf{choose}^{\star}[\top]\langle \mathsf{my}.nodeL\rangle.Advert$$

$$Arrive \stackrel{\text{def}}{=} \mathsf{arrive}^{\star}[\mathsf{my}.nodeL = arrL](arrL, startTime)\ \{\mathsf{my}.arrived \leftarrow \mathsf{my}.arrived + 1, timeSum := timeSum + (now - startTime)\}.Arrive$$

$$StartAssign \stackrel{\text{def}}{=} \mathsf{assignStart}^{\star}[\bot]\langle \mathsf{my}.nodeL\rangle.StartAssign$$

$$GoalAssign \stackrel{\text{def}}{=} \mathsf{assignGoal}^{\star}[\bot]\langle \mathsf{my}.nodeL\rangle.GoalAssign$$

Initial state of *PathNode* component: $Advert \mid Arrival \mid StartAssign \mid GoalAssign$

Fig. 17. The *PathNode* component of the Meadows model

6.2 Results

We simulated the model for the following two cases:

(1) all the lanes are available to the Pedestrians,
(2) one of the segments (the middle segment of the Middle Meadow Walk, the edge between nodes 1 and 4) is closed, for example because of having been flooded.

The scenarios described above are two possible example situations to explore. If required, the model can be easily adapted to different path closure situations.

The results presented in Fig. **??** are dependant on the shape of the network of paths as well as the constants in the formulas. At any node a Pedestrian has to make a decision, which in our system is modelled by the stochastic nature of Carma actions. The outcome depends on how congested the potential lanes are, as well as on how optimal the next node is in terms of its distance from the goal location. Varying the weights of these two factors can have an influence on the behaviour in the system.

For each of the two scenarios we obtained snapshots of the state of the systems at four times of the day: $t \in [9, 13, 17, 21]$ (hours). The snapshots presented in Fig. **??**–**??** show average amounts of pedestrians on graph edges measured in the time period $(t - \delta, t + \delta)$, where $\delta = 1$hour. The two colours of arrows denote two opposite directions of the movement of pedestrians. The length of an arrow represents the proportion of pedestrians travelling in one direction and pedestrians travelling in the other direction to the total number of pedestrians on that edge. The total number of pedestrians on a given edge is represented by the arrow's thickness.

The result show that in the initial scenario, a large portion of pedestrians choose to travel along the Middle Meadow Walk. In the second scenario, the closure of the central segment of The Middle Meadow Walk forces pedestrians to choose a different path instead. This increases the traffic on the nearby lanes, however the lanes that are farther away are not significantly affected, as the traffic disperses through the network more evenly after a number of movement steps.

Space:

Nodes	set of coordinate pairs representing nodes in the graph
Connections	set of node pairs representing edges in the graph
Distances	a two-dimensional lookup table of pre-computed distances between pairs of nodes

Measures:

$traffic(n_1, n_2)$	the number of pedestrians whose current location is n_1 and next location is n_2
$arrived(n)$	the number of pedestrians who finished their route at the node n

Evaluation context:

$$\mu_p(\gamma_s, \gamma_r, \alpha) = \begin{cases} \mathbf{Pr_{goal}}\big(\gamma_s(nodeL), now\big) & \text{if } \alpha = \mathsf{assignGoal}^\star \\ \mathbf{Pr_{start}}\big(\gamma_s(nodeL), now\big) & \text{if } \alpha = \mathsf{assignStart}^\star \\ \mathbf{Pr_{choose}}\big(\gamma_s(nodeL), \gamma_r(currL), \gamma_r(nextL).goal, & \\ \qquad traffic(\gamma_r(nextL), \gamma_r(currL)\big) & \text{if } \alpha = \mathsf{choose}^\star \\ \mathbf{1} & \text{otherwise} \end{cases}$$

$$\mu_w(\gamma_s, \gamma_r, \alpha) = 1$$

$$\mu_r(\gamma_s, \alpha) = \begin{cases} \mathbf{Rate_{spawn}}\big(now\big) & \text{if } \alpha = \mathsf{spawn}^\star \\ \mathbf{Rate_{move}}\big(\gamma_s(currL), \gamma_s(nextL), & \\ \qquad traffic(\gamma_s(nextL), \gamma_s(currL)\big) & \text{if } \alpha = \mathsf{choose}^\star \\ \lambda_{fast} & \text{otherwise} \end{cases}$$

$$\mu_u(\gamma_s, \alpha) = \begin{cases} \emptyset, \big(Pedestrian, \{startL \leftarrow \gamma_s(startL), goalL \leftarrow \gamma_s(goalL), stime \leftarrow \mathsf{now}\}\big) & \\ \qquad \text{if } \alpha = \mathsf{spawn}^\star & \\ \{\}, 0 \quad \text{otherwise} & \end{cases}$$

Initial collective:

$$Meadows \stackrel{\text{def}}{=} \big(Generator, \emptyset\big) \parallel \big(PathNode, \{\, nodeL \mapsto Nodes_1 \,\}\big) \parallel \ldots \parallel \big(PathNode, \{\, nodeL \mapsto Nodes_p \,\}\big)$$

Fig. 18. Environment and evaluation context of the Meadows model

Developing and analysing this model illustrates the usage of the CARMA language and its software suite for the purpose of developing models based on real data. At present, the only data on the Meadows city park available for analysis is the geographic data from the Google Maps API and the bike count data from the City Council, measured at a single point in the park. If real world data measuring pedestrian traffic at a number of locations around the park becomes available, it may be used to parametrise the presented model, which would allow us to evaluate the predictive power of the model.

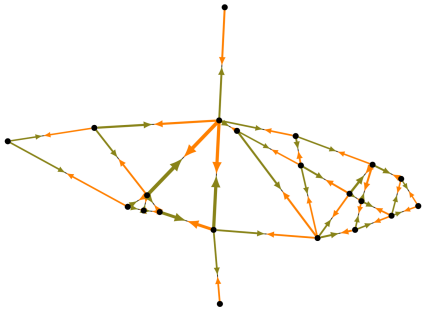


Fig. 19. Scenario (1) at time $t = 9$

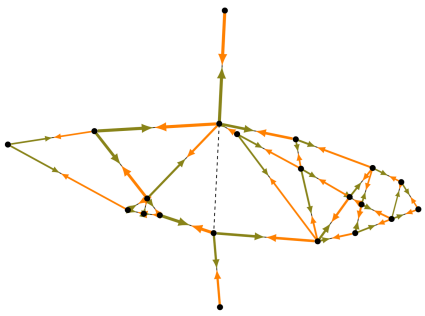


Fig. 20. Scenario (2) at time $t = 9$

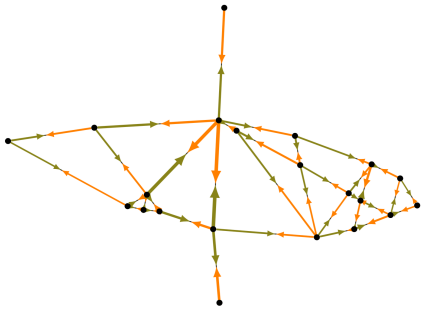


Fig. 21. Scenario (1) at time $t = 13$

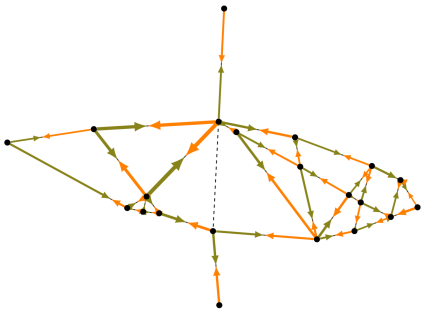


Fig. 22. Scenario (2) at time $t = 13$

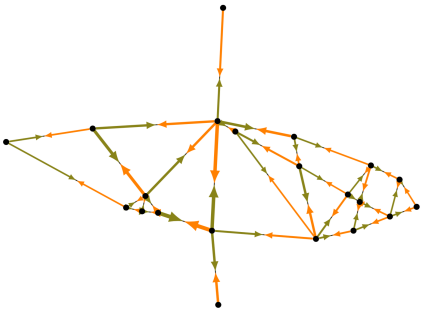


Fig. 23. Scenario (1) at time $t = 17$

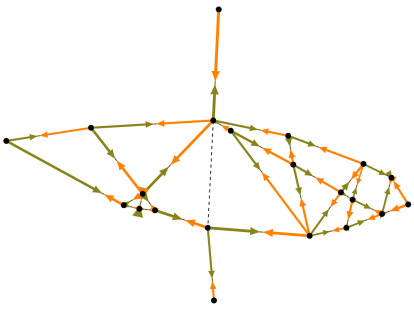


Fig. 24. Scenario (2) at time $t = 17$

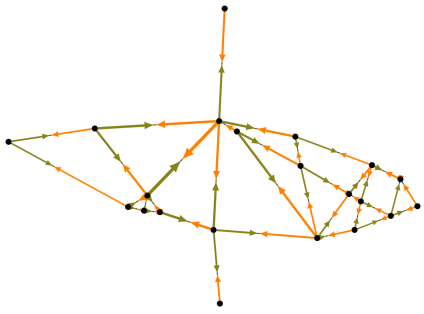


Fig. 25. Scenario (1) at time $t = 21$

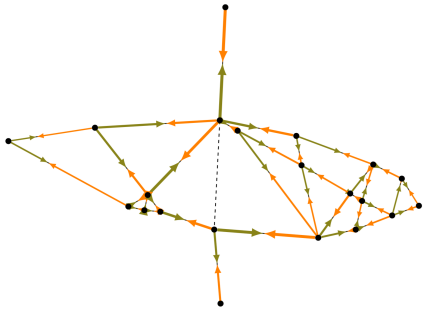


Fig. 26. Scenario (2) at time $t = 21$

7 CONCLUSIONS

In this paper, we have formulated a mesoscopic model of a system of pedestrian movement as a collective adaptive system where the actions of some participants within the system are influenced by the actions (or even the presence) of others. We approached the creation of a more detailed and more realistic model by building on the experience which we gained in creating a basic model with simple and regular symmetric structure. The insights obtained from this basic model directed our attention to likely issues with the more realistic model, and deepened our understanding of the effects of local decisions on global behaviour.

The degree of self-awareness in the system gained through the (local) recognition and attempted avoidance of congestion in the network by the participants within the system itself makes the problem different in character from a traditional network-flow problem where data is routed through a network without even local knowledge of congestion ahead. Adaptive systems naturally give rise to phenomena such as these where the decisions of one participant are influenced by the decisions of another seemingly entirely independent participant and the longer-term consequences of a decision are usually infeasible to predict at the time of making the decision, and it may even be difficult to understand the path from decision to consequence in retrospect.

The suite of experiments which we carried out on the models presented here, and the many variants of these which were created in the model development process, provided us with a convincing demonstration of the applicability of the Carma modelling tools to a mesoscopic modelling problem and evidence of the practical effectiveness of the Carma process calculus as an intellectual vehicle for expressing intelligent density-dependent movement across an arbitrary network topology.

We made extensive use of the CaSL language, which enriches the core Carma process calculus with additional language features thereby making it possible to express large models cleanly by using data types and data structures to represent structured information in the model, allowing the model to be statically checked for model coherence at compile-time. This eliminated a raft of potential modelling errors, streamlining the process of model creation. CaSL additionally supports arbitrary function definitions for the propensity functions which give rise to probability distributions over model actions, allowing us to model user behaviour accurately; and general rate functions, allowing us to model timing behaviour accurately. In addition, it provides spatial data structures, allowing us to represent physical separation and distance accurately.

Taken as a modelling exercise, we have found our creation of a model of pedestrian movement on the Meadows to be a valuable one. The consequences of network separation and path loss are not easy to see and the results from our model analysis show the surprising effects of even small changes to the network structure. Our development of a detailed spatial model of the system, parameterised by up-to-date user measurement data, gives us a precise formal means of investigating even compound changes to the network, and presenting the results of the analysis back to system stakeholders in a form which will enable them to take well-informed planning decisions, supported by objective analysis evidence.

ACKNOWLEDGMENTS

This work is supported by the EU project QUANTICOL, 600708. We thank Michele Loreti for many helpful suggestions on our models and for assisting us with using the Carma tools. We thank the anonymous reviewers of the FORECAST workshop for many helpful comments which assisted us in improving an earlier version of this paper.

Received February 2017; revised August 2017; accepted October 2017